AF 2132 $ 120

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants: **JÜRGEN REINOLD, ET AL.**

Serial No.: 09/944,893

Title: Vehicle Active Network with Data Encryption

Filed: August 31, 2001

Group Art Unit: 2132

Examiner: Cas P. Stulberger

I hereby certify that this paper is being deposited with the United States Postal Service as first-class mail, postage prepaid, in an envelope addressed to:

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

on this date **October 18, 2004**

Anthony G. Sitko
Reg. No. 36,278
Attorney for Applicant & Assignee

APPEAL BRIEF

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Pursuant to the Notice of Appeal mailed May 18, 2004 in connection with the above-identified patent application, the applicants respectfully submit the instant Brief on Appeal in accordance with 37 C.F.R. § 1.192. Enclosed is a check in the amount of $340.00, pursuant to 37 C.F.R. § 1.17(c). If there are any additional fees or refunds required, the Commissioner is directed to charge or debit Deposit Account No. 13-2855. The applicants further enclose a Petition to extend the time for filing this brief until October 18, 2004.

I. REAL PARTY IN INTEREST

The real party in interest is Motorola, Inc. the assignee of the above-identified patent application. The assignment assigning rights to Motorola, Inc., is recorded in the United States Patent and Trademark Office ("USPTO") at Frame 0047 of Reel 012563.

II. RELATED APPEALS AND INTERFERENCES

There are no related interferences.

An appeal against the rejection of each of the following related patent applications has been filed:

Serial number 09/943,882 entitled VEHICLE ACTIVE NETWORK WITH FAULT TOLERANT DEVICES;

Serial number 09/943,921 entitled VEHICLE ACTIVE NETWORK WITH BACKBONE STRUCTURE; and

Serial number 09/944,892 entitled VEHICLE ACTIVE NETWORK WITH RESERVED PORTIONS.

III. STATUS OF THE CLAIMS

Currently, claims 1-15 are pending in the application. The pending claims are presented in Appendix A to this Brief. Claims 1-15 stand rejected and form the subject matter of this appeal

The application was filed on August 31, 2001, with claims 1-15.

A preliminary amendment filed on May 23, 2002 made minor changes to the specification to include reference to related applications.

The first Office action mailed July 30, 2003, *inter alia*, rejected claims 1-4, 6 and 8-15 under 35 U.S.C. § 103(a) as unpatentable over Pogue Jr. (U.S. Patent No. 5,995,512) in view of Daniels et al (US Patent No. 5,991,401). The examiner alleged Pogue Jr. disclosed within the preferred environment of an automobile a data network capable of transmitting audio, video, data, low-bandwidth control date, and other similar signals. First and second device limitations are alleged to be met by the disclosure of a CD player and an audio processor/amplifier coupled to the network. The examiner noted that Pogue Jr. does not disclose data encryption, use of data

encryption for error detection, a bridge, a router or a switch. The examiner further alleged, however, that Daniels et al discloses a network in which individual data packet are encrypted and decrypted, rejection of packets potentially harmful to the network and a PCI host bridge.

With respect to claims 5 and 7, the combination of Pogue Jr. and Daniels et al failing to disclose or suggest a router or a switch, such elements are allegedly found in Wright et al (US Patent No. 6,101,599).

Thus, the first Office action concludes that claims 1-15 fail to meet the requirements of patentability set forth in 35 U.S.C. § 103(a).

The applicants responded to the Office action on August 27, 2003 arguing that the examiner failed to make out a *prima facie* case of obviousness because there is no motivation in the references themselves or in the art to combine of Pogue Jr. and Daniels et al and/or Pogue Jr., Daniels et al and Wright et al, and that even is such motivation is found, the combination fails to disclose or suggest the claimed invention.

In particular, the combinations do not disclose or even suggest a vehicle comprising a first device, a second device and an active network communicatively coupling the first and second devices. Furthermore, the combination discloses a star topology data bus utilizing a master controller, (Pogue Jr., Fig. 1, abstract and col. 7, lines 62-67) or a passive packet network (Daniels et al) and does not disclose or teach an active network, which is fundamentally different. The applicants argued an active network is a network in which the nodes (active elements) are programmed to perform custom operations on the messages that pass through the node; thus active elements within an active network enable multiple simultaneous communication paths between devices within the network (page 7, lines 6-7 of Applicant's specification). This is in stark contrast to the passive network-types disclosed by the combinations wherein the

network is only aware of the destination of the messages passing through the network nodes and are designed to deliver exactly one substantially unmodified copy of the message to the ultimate destination.

The examiner issued a final Office action on December 18, 2004, rejecting claims 1-15 in view of the cited combinations and for the very same reasons as set forth in the first Office action. The examiner responded to the applicants arguments that the claims require an active network having a particular structure and functionality not taught by the combinations by stating that the applicants were arguing limitations not found in the claims.

The applicants response to the final Office action filed on April 19, 2004 contained no new amendments. The applicants maintained that there is no motivation to form the alleged combinations and that the combinations fail to disclose or suggest the claimed invention including, *inter alia*, methods of data communication in a vehicle including an active network. The applicants asserted that an active network is known to the skilled artisan to include nodes capable of performing custom operations on the messages that pass through the nodes. An active network does not require a central server or computing resources. And, active network nodes are aware of the contents of messages transported and can participate in the processing and modification of the messages while they travel through the network. Applicants further argued that the broadest reasonable interpretation of the term "active network" in the claims must include an active network with the above-stated characteristics because it is both consistent with the specification and with what one of ordinary skill in the art would understand the term to mean.

The applicants argued that because Pogue Jr. does not teach or suggest an active network and Daniels et al does not teach or suggest an active network, the combination

of Pogue Jr. in view of Daniels et al does not teach or suggest an active network regardless of what other structures or functions these references may teach or suggest. Furthermore, because Wright et al also fail to disclose or suggest an active network, the combination of Pogue Jr., Daniels et al and Wright et al does not teach or suggest each and every limitation contained in the claims. Therefore, the applicants asserted, the claims 1-15 meet the requirements for patentability and are allowable.

An advisory action issued May 14, 2004 maintaining the rejection of the claims, and this appeal followed.

IV. SUMMARY OF THE INVENTION

Although specification citations are inserted below in accordance with C.F.R. 1.192(c), these reference numerals and citations are merely examples of where support may be found in the specification for the terms used in this section of the brief. There is no intention to in any way suggest that the terms of the claims are limited to the examples in the specification. Although, as demonstrated by the reference numerals and citations below, the claims are fully supported by the specification as required by law, it is improper under the law to read limitations from the specification into the claims. Pointing out specification support for the claim terminology, as is done here to comply with rule 1.192(c), does not in any way limit the scope of the claims to those examples from which they find support. Nor does this exercise provide a mechanism for circumventing the law precluding reading limitations into the claims from the specification. In short, the reference numerals and specification citations are not to be construed as claim limitations or in any way used to limit the scope of the claims.

The invention, as defined in claims 1 and 11 and with reference to FIGS. 1-4, is a vehicle 10 including a first device, e.g., 14, 16, 18, 20, 46 or 48 and second device, e.g., 14, 16, 18, 20, 46 or 48 and an active network 30 communicatively coupling the

first device and the second device. Data, and particularly data packets communicated inter-network and intra-network, as the case may be, may be encrypted. Page 17, lines 18-22 and page 22, lines 7-18.

V. ISSUES ON APPEAL

The issue presented on appeal is: are each of pending claims 1-15 patentable over the combination of Pogue Jr. (U.S. Patent No. 5,499,247) in view of Daniels et al (US Patent No. 5,940,372) and/or the combination of Pogue Jr. (U.S. Patent No. 5,499,247) in view of Daniels et al (US Patent No. 5,940,372) and in further view of Wright et al (US Patent No. 6,101,599).

VI. ARGUMENT

In the final Office action dated December 18, 2003, the examiner does not seem to dispute the applicants' argument that none of the cited references disclose an active network or that there exists a motivation in the references or the art to modify the existing network architectures as active network architectures. Instead, the examiner's response is that the applicants argue structure and limitations that are not present in the claims. However, the applicants use the term "active network" as a noun. Thus, the structure the examiner argues is absent is contained in the term itself.

First, it is useful to understand what the applicants mean by the term "active network." Those of ordinary skill in the art know perfectly what an active network is, what an active network does and how to realize an active network. This is not an arbitrary assumption made by the attorney, but is based upon experts in the field of computing and networking. See references and articles attached in Appendices B, C and D. These additional references and articles, written by third parties, demonstrate that an active network is a name used as a noun for recognizing a very particular kind of network.

Owing to the fact that applicant does not provide a special definition of the term "active network", such term must be given its plain meaning, i.e. it must be read as it would be interpreted by those of ordinary skill in the art. In any case, the broadest reasonable interpretation must to consistent with the specification and must also be consistent with the interpretation that those skilled in the art would reach. *See* MPEP § 2111.01: "during examination the pending claims must be given their broadest reasonable interpretation consistent with the specification ... the broadest reasonable interpretation of the claims must also be consistent with the interpretation that those skilled in the art would reach ...." The interpretation of the term "active network" given by those of ordinary skill in the art is clear (see the aforementioned attached references): an active network is a network including nodes capable of performing custom operations on the messages that pass through the nodes; does not require a central server or computing resource; are aware of the contents of the messages transported and can participate in the processing and modification of the message while they travel through the network.

A. The references do not teach an active network

Alleged by the examiner is that Pogue Jr. "discloses ... an active network." The network is not an active network. Only if one considers the term "active" as a simple adjective to the word "network"; in this light, an "active network" is a network capable of doing any kind of action, does it follow that the network of Pogue Jr. is an active network. Following such an interpretation, however, every network is an active network, due to the fact that every network is at least able of establishing a connection. Thus it is impossible to claim a network which is not active with this meaning (a non-active network must be a network which does not do anything, and thus it is a

completely unuseful network), this interpretation leads to the word "active" conferring no kind of limitation to the word "network".

What Pogue Jr. does clearly disclose, however, is that the network is a bus arranged in a star topology, and that the network includes a master controller (Fig. 1, abstract, col. 7, lines 62-67). If we try to give to the term "active network" the aforementioned meaning given by the examiner, i.e., a star topology bus with a master controller is an active network, it is clear that to do so gives a meaning that is not consistent with the interpretation that those skilled in the art would reach with respect to the term active network. Furthermore, we are giving to the term "active network" a meaning that is not consistent with the specification: according to this meaning, e.g., a bus network is an active network, but in the specification it is clearly stated that a bus network is not an active network. Both such considerations demonstrate without any doubt that this interpretation does not conform with what one of ordinary skill in the art would understand an active network to be and is inconsistent with the teachings of the specification.

Under similar reasoning, the bus-type network of Daniels et al. and Wright et al. also are not active networks, as that term is used in the claims.

B. The combination of Pogue Jr. and Daniels et al and/or Pogue Jr. and Daniels et al in further view of Wright et al do not render the claimed invention unpatentable

Knowing that references do not teach an active network, not only must that teaching be found elsewhere, but to establish a *prima facie* case of obviousness, and hence to find the claims 1-15 unpatentable under 35 U.S.C. § 103(a), three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there

must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all of the claim limitations. The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, and not be based upon applicant's disclosure. MPEP at § 2142.

As discussed above, the applicants contend that none of the cited references teach an active network. Hence it follows that any combination of the references, assuming *arguendo* motivation to make such a combination, does not teach an active network, a vehicle incorporating an active network or an active network with data encryption. Hence, the claims 1-15 are patentable over the combination.

Notwithstanding that the references fail to teach an active network, the applicants admit that the term active network describes a known network type. *See* Appendices B, C and D. What is not taught or suggested in the art, and what the art does not establish, is a suggestion or motivation to use an active network in a vehicle. That comes only from the applicants' own specification, and to conclude such is inappropriate hindsight.

Careful analysis of the cited references reveals no suggestion or motivation to modify or replace a network used in a vehicle with an active network. There is no suggestion of any deficiency in the network disclosed by Pogue Jr. that would be overcome by the use of an active network or any benefit to be gained by using an active network. Neither Daniels et al nor Wright et al is directed to vehicle applications so that even if they had disclosed an active network one would <u>not</u> be motivated to incorporate the disclosed active network architecture in a vehicle. Nor do the references cited by the applicants suggest use of an active network in a vehicle. It is only by the applicants' disclosure is one first taught to make the combination of a

vehicle and an active network. The examiner has failed to point to the motivation or suggestion contained within the references or the art for making the modification or combination. MPEP § 2142.

Because there is no suggestion or motivation in the references themselves to combine a vehicle and an active network, it follows that claims 1-15 are patentable.

C. Conclusion

Clear from the foregoing discussion, the applicants have claimed a specific physical structure, namely an active network known to have particular characteristics, within a vehicle. This active network is not a bus architecture and is not a passive network or a combination of a passive network and a bus architecture or any other type of network structure than an active network structure. In light of the specification, the broadest reasonable interpretation of the term active network does not include bus structures and/or passive networks. For the claims to be unpatentable, i.e., not to meet the requirements of § 103(a), the prior art must teach or suggest each and every limitation contained in the claims as well as to provide the motivation or suggestion to combine the references, and particularly, in this case, must teach or suggest making a vehicle including an active network. Because the prior art fails to teach or suggest this structure or methods employing such structures, claims 1-15 do meet the requirements of 35 U.S.C. § 103(a) and are patentable.

Respectfully submitted,

October 18, 2004 By:

Anthony G. Sitko
Attorney for Applicant & Assignee
Registration No. 36,278
Marshall, Gerstein & Borun LLP
233 South Wacker Drive, 6300 Sears Tower
Chicago, Illinois 60606-6357
Tel.: (312) 474-6300
Fax: (312) 474-0448

APPENDIX A

CLAIMS

1. (original) In a vehicle comprising a first device and a second device and an active network communicatively coupling the first device and the second device for the communication of data between the first device and the second device, the active network being operable to encrypt the data.

2. (original) The vehicle of claim 1, wherein each of the first device and the second device is coupled via an interface to the active network, and wherein each interface is operable to encrypt and decrypt the data.

3. (original) The vehicle of claim 1, wherein the active network comprises a plurality of active network elements coupled by connection media.

4. (original) The vehicle of claim 3, wherein at least one of the plurality active network elements is operable to encrypt and decrypt the data.

5. (original) The vehicle of claim 3, wherein at least one of active network elements comprises a switch.

6. (original) The vehicle of claim 3, wherein at least one of active network elements comprises a bridge.

7. (original) The vehicle of claim 3, wherein at least one of active network elements comprises a router.

8. (original) The vehicle of claim 1, wherein the active network is operable to determine an error in the data based upon the encryption of the data.

9. (original) The vehicle of claim 1, wherein the data comprises data packets, and wherein the active network is operable to encrypt the data packets.

10. (original) The vehicle of claim 9, wherein the data packets are individually encrypted.

11. (original) A method of communicating data between a first device and a second device within a vehicle, the vehicle including an active network communicatively coupling the first device and the second device, the method comprising the steps of:

receiving data from the first device to be communicated to the second device via the active network;

encrypting the data at a first interface, the first interface coupling the first device to the active network,

communicating the data to a second interface, the second interface coupling the second device to the active network,

decrypting the data at the second interface; and

communicating the data to the second device.

12. (original) The method of claim 11, wherein the first interface and the second interface each comprise active network elements of the active network.

13. (original) The method of claim 11, further comprising detecting an error in the data at the second interface.

14. (original) The method of claim 13, wherein the step of detecting an error in the data comprises detecting an error in the data based upon the encryption.

15. (original) The method of claim 11, wherein the data comprise data packets, and wherein the step of encrypting the data comprises encrypting the data packets and wherein the step of decrypting the data comprises decrypting the data packets.

Towards an Active Network Architecture

David L. Tennenhouse and David J. Wetherall*
Telemedia, Networks and Systems Group, MIT

ABSTRACT

Active networks allow their users to inject customized programs into the nodes of the network. An extreme case, in which we are most interested, replaces packets with "capsules" – program fragments that are executed at each network router/switch they traverse.

Active architectures permit a massive increase in the sophistication of the computation that is performed within the network. They will enable new applications, especially those based on application-specific multicast, information fusion, and other services that leverage network-based computation and storage. Furthermore, they will accelerate the pace of innovation by decoupling network services from the underlying hardware and allowing new services to be loaded into the infrastructure on demand.

In this paper, we describe our vision of an active network architecture, outline our approach to its design, and survey the technologies that can be brought to bear on its implementation. We propose that the research community mount a joint effort to develop and deploy a wide area ActiveNet.

1. INTRODUCTION

Traditional data networks passively transport bits from one end system to another. Ideally, the user data is transferred opaquely, i.e., the network is insensitive to the bits it carries and they are transferred between end systems without modification. The role of computation within such networks is extremely limited, e.g., header processing in packet-switched networks and signaling in connection-oriented networks.

Active Networks break with tradition by allowing the network to perform customized computations on the user data. For example, a user of an active network could send a customized compression program to a node within the network (e.g., a router) and request that the node execute that program when processing their packets. These networks are "active" in two ways:

- Switches perform computations on the user data flowing through them.
- Individuals can inject programs into the network, thereby tailoring the node processing to be user- and application-specific.

We have identified several architectural approaches to active networks. One approach, which we find particularly interesting, replaces the passive packets of present day architectures with active "capsules" – miniature programs that are executed at each router they traverse. This change in architectural perspective, from passive packets to active capsules, simultaneously addresses both of the "active" properties described above. User data can be embedded within these mini-programs, in much the way a page's contents are embedded within a fragment of PostScript code. Furthermore, capsules can invoke pre-defined program methods or plant new ones within network nodes.

Our work is motivated by both technology "push" and user "pull". The technology "push" is the emergence of "active" technologies, compiled and interpreted, supporting the encapsulation, transfer, interposition, and safe and efficient execution of program fragments. Today, active technologies are applied within individual end systems and above the end-to-end network layer; for example, to allow Web servers and clients to exchange program fragments. Our innovation is to leverage and extend these technologies for use within the network – in ways that will fundamentally change today's model of what is "in" the network.

The "pull" comes from the ad hoc collection of firewalls, Web proxies, multicast routers, mobile proxies, video gateways, etc. that perform user-driven computation at nodes "within" the network. Despite architectural injunctions against them, these nodes are flourishing, suggesting user and management demand for their services. We are developing the architectural support and common programming platforms to support the diversity and dynamic deployment requirements of these "interposed" services. Our goal is to replace the numerous ad hoc approaches to their implementation with a generic capability that allows users to program their networks.

There are three principal advantages to basing the network architecture on the exchange of active programs, rather than passive packets:

- Exchanging code provides a basis for adaptive protocols, enabling richer interactions than the exchange of fixed data formats.
- Capsules provide a means of implementing fine grained application-specific functions at strategic points within the network.

*{dlt,djw}@lcs.mit.edu. http://www.tns.lcs.mit.edu/. An earlier version of this paper was delivered during the keynote session of Multimedia Computing and Networking, San Jose, CA, January 1996.

- The programming abstraction provides a powerful platform for user-driven customization of the infrastructure, allowing new services to be deployed at a faster pace than can be sustained by vendor driven standardization processes.

This paper presents our vision of an active network architecture and the approach we are following towards the deployment of an operational ActiveNet. The active network approach opens a Pandora's box of safety, security, and resource allocation issues. Although we do not present a complete design, we identify a number of specific research issues, outline the approach we are following towards their resolution and identify the technologies we intend to leverage. Our plan is to bootstrap a wide area ActiveNet using similar techniques to those used by the prototype MBONE, i.e., by locating platforms at strategic locations and "tunneling" through existing transmission facilities, such as the Internet.

In the next section we provide a description of some of the "lead user" applications that motivate an architecture that facilitates computation within the network. In section 3, we provide an overview of active networks, a high-level perspective on how we propose to organize their platforms and an introduction to the research issues that must be addressed. Section 4 describes the "instruction set" issues associated with an interoperable programming model and how "active technologies" can be leveraged to effect the safe and efficient evaluation of capsules. We then discuss the management of node resources, such as storage and link bandwidth, followed by our plan for the deployment of a research ActiveNet. We realize that our work challenges some key assumptions that have guided recent networking research and so the final sections of this paper discuss the architectural and structural questions raised by our approach.

2. LEAD USERS

Recently, there has been considerable interest in: agent technologies, which allow mobile code to travel from clients to servers; and in Web applets, which allow mobile code to travel from servers to clients. Active networks bridge this dichotomy by allowing applications to dispatch computation to where it is needed.

We are encouraged by the observation that a number of lead users have pressing requirements for the transparent interposition of computation within the network. These include the developers of:

- Firewalls, which are typically located at administrative boundaries.
- Web proxies and other services, such as DNS and multicast routers, that form strategic vertices of copy, fusion and cache "trees".
- Mobile/Nomadic gateways, placed near the edges of the network where there are significant discontinuities in the available bandwidth, e.g., the base stations of wireless networks.

These lead applications demonstrate that there is user "pull" towards active networks. In the absence of a coherent approach to interposition they have adopted a variety of ad hoc strategies. In many cases the interposed platforms present the facade of network layer routers, but actually perform application- or user-specific functions. Active networks will rationalize these diverse activities by providing a uniform platform for network-based computation.

Firewalls

Firewalls implement filters that determine which packets should be passed transparently and which should be blocked. Although they have a peer relationship to other routers, they implement application- and user- specific functions, in addition to packet routing. The need to update the firewall to enable the use of new applications is an impediment to their adoption. In an Active Network, this process could be automated by allowing applications from approved vendors to authenticate themselves to the firewall and inject the appropriate modules into it.

Web Proxies

Web proxies are an example of an application-specific service that is tailored to the serving and caching of World Wide Web pages. Harvest [1] employs a hierarchical scheme in which cache nodes are located near the edges of the network, i.e., within the end user organizations. This system is scalable and could be extended by allowing nodes of the hierarchy to be located at strategic points within the networks of the access providers and inter-exchange carriers. An interesting problem is the development of algorithms and tools that automatically balance the hierarchy by re-positioning the caches themselves, not just the cached information. Schemes such as dynamic hierarchical caching [2] and geographical push-caching [3] begin to address this issue.

A further argument in favor of using active technologies for web caching is that a significant fraction of web pages are dynamically computed and not susceptible to traditional (passive) caching. This suggests the development of web proxy schemes that support "active" caches that store and execute the programs that generate web pages.

Mobile/Nomadic Computing

Interposition strategies are used by a number of researchers addressing mobility. For example, Kleinrock [4] describes a "nomadic router" that is interposed between an end system and the network. This module observes and adapts to the means by which the end system is connected to the network, e.g., through a phone line in a hotel room versus through the LAN in the home office. It might decide to perform more file caching or link compression when the end system is connected through a low bandwidth link

and/or invoke additional security, such as encryption, when operating away from the home office.

Similarly, "nomadic agents and gateways" [4] are nodes that support mobility. They are located at strategic points that bridge networks with vastly different bandwidth and reliability characteristics, such as the junctions between wired and wireless networks. Application-neutral work on TCP snooping [5] improves the performance of TCP connections by retaining per-connection state information at wireless base stations. Application-specific services performed at gateways include file caching and the transcoding of images [6]. The InfoPad [7] takes the process even further, by instantiating user-specific "pad servers" supporting a range of applications, such as voice and hand-writing recognition, at intermediate nodes.

New Application Domains

There is an untapped reservoir of applications that require sophisticated network-based services to support the distribution and fusion of information. One promising direction is the development of multi-point communication strategies that are more flexible than the existing IP multicast service, which performs a very limited computation on the user data, i.e., copying. Application-specific multicast, for example, would provide the mechanism to realize the quality of service filtering suggested in [8] for video-conferencing.

Information fusion is an example of a domain that may leverage interposed computation. Applications such as sensor fusion, simulation and remote manipulation, allow users to "see" composite images constructed by fusing information obtained from a number of sensors. Fusing data within the network reduces the bandwidth requirements at the users, who are located at the periphery of the network. Placing application-specific computation near where it is needed also addresses latency limitations by shortening the critical feedback loops of interactive applications.

3. ACTIVE NETWORKS

In this section, we provide an overview of active networks – highly programmable networks that perform computations on the user data that is passing through them. We distinguish two approaches to active networks, discrete and integrated, depending on whether programs and data are carried discretely, i.e., within separate messages, or in an integrated fashion. We then provide a high-level description of how active nodes might be organized and describe a node programming model that could provide the basis for cross-platform interoperability.

3.1 Programmable Switches – A Discrete Approach

The processing of messages may be architecturally separated from the business of injecting programs into the node, with a separate mechanism for each function. Users would send their packets through such a "programmable" node much the way they do today. When a packet arrives, its header is examined and a program is dispatched to operate on its contents. The program actively processes the packet, possibly changing its contents. A degree of customized computation is possible because the header of the message identifies which program should be run – so it is possible to arrange for different programs to be executed for different users or applications.

The separation of program execution and loading might be valuable when it is desirable for program loading to be carefully controlled or when the individual programs are relatively large. This approach is used, for example, in the Intelligent Network being standardized by CCITT. In the Internet, program loading could be restricted to a router's operator who is furnished with a "back door" through which they can dynamically load code. This back door would at minimum authenticate the operator and might also perform extensive checks on the code that is being loaded. Note that allowing operators to dynamically load code into their routers would be useful for router extensibility purposes, even if the programs do not perform application- or user-specific computations.

3.2 Capsules – An Integrated Approach

A more extreme view of active networks is one in which every message is a program. Every message, or capsule, that passes between nodes contains a program fragment (of at least one instruction) that may include embedded data. When a capsule arrives at an active node, its contents are evaluated, in much the same way that a PostScript printer interprets the contents of each file that is sent to it.

Figure 1 provides a conceptual view of how an active node might be organized. Bits arriving on incoming links are processed by a mechanism that identifies capsule boundaries, possibly using the framing mechanisms provided by traditional link layer protocols. The capsule's contents are dispatched to a transient execution environment where they can safely be evaluated. We hypothesize that programs are composed of "primitive" instructions, that perform basic computations on the capsule contents, and can also invoke external "methods", which may provide access to resources external to the transient environment. The execution of a capsule results in the scheduling of zero or more capsules for transmission on the outgoing links and may change the non-transient state of the node. The transient environment is destroyed when capsule evaluation terminates.

3.3 Programming With Capsules

Our distinction between the discrete and integrated approaches is one of perspective, primarily useful as a basis for comparing two ways of thinking about networks and their programming. In practical terms, a network based on the integrated approach could be programmed to emulate the discrete approach and vice-versa. Nonetheless, we are intrigued by the

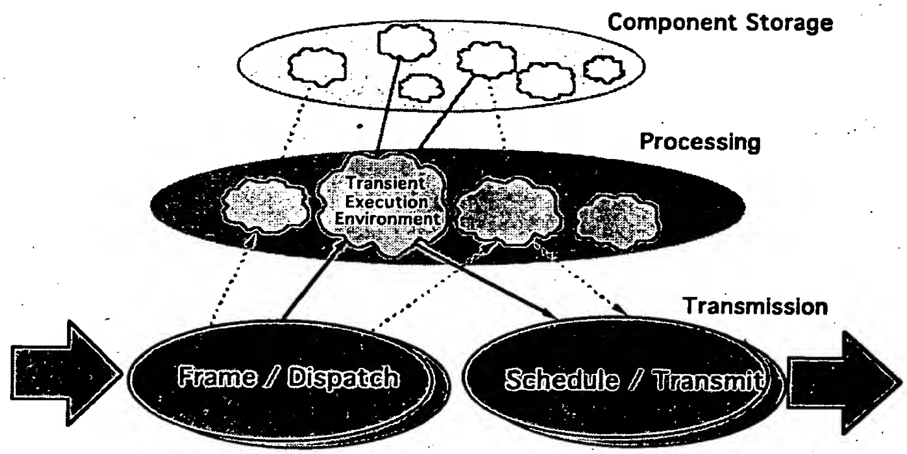


Figure 1. Active Node Organization

possibilities afforded by the integrated perspective, especially with respect to new ways of leveraging computation within the network. It provides a programming language framework for thinking about networks – a framework that could enable the synthesis of recent results in the areas of programming environments, operating systems and networks.

In the following paragraphs we discuss some ways in which active networks could be leveraged to support a variety of traditional functions, such as IP packet processing, connections, flows, routing protocols, etc. These examples are meant to provide insight into the "flavor" of the networks we envision and establish the groundwork for the discussion of the programming model and implementation technologies which follows.

In simple applications, a capsule's actions on visiting a node are to compute its "next hop" destination(s) and schedule zero or more (possibly modified) copies of itself for transmission on selected links. It will be necessary to provide mechanisms for determining and naming the links on which outgoing capsules are transmitted. In the IP protocol, this mechanism is "built-in" to every node and individual packets need only carry their destination address – they need not have knowledge of the links they traverse. In pure source routing schemes, each message carries the identities of all of the links it traverses. We hope to develop an intermediate approach, in which capsules can dynamically enumerate and evaluate the paths available at a node, without requiring detailed knowledge at the time the capsule is composed.

An important question concerns the degree to which a capsule program can access objects, such as routing tables, that lie beyond the transient execution environment. In a restricted approach, capsules could be largely self-contained. Although sufficient to implement some interesting programs, e.g., the above-mentioned source routing, this model is somewhat confining. In the following paragraphs we discuss three ways in which programs could reach beyond the capsule's transient environment:

- Foundation Components – universally available services implemented outside of the capsule.
- Active Storage – the ability to modify the state that node storage is left in at the completion of capsule execution.
- Extensibility – allowing programs to define new classes and methods.

Foundation Components

Foundation components implement external "methods" that provide controlled access to resources outside of the transient execution environment. A subset of these components will reflect the "API" of the node's run-time environment to the applications. Other components provide a built-in class hierarchy that serves as a base for the development of capsule programs.

Many capsules will require access to other node-specific information and services, such as routing tables and the state of the node's transmission links. Using built-in components that provide access to this information, one could design capsules whose evaluation performs similar processing to that performed on the header of an IP datagram. Multi-cast and option processing instructions could be included in the capsules that require them. Whereas the traditional IP approach calls for the code to be fixed and built into the router, in the active case the program is flexible and carried with the data.

For migration purposes, we could develop standardized components that implement the existing Internet protocol types. A capsule carrying an embedded IPv4 datagram could contain a single instruction of the form "execute the IPv4 method on the remainder of the payload". To put matters in perspective, we can think of existing routers as an

extremely restricted subset of active nodes, in which the capsule "program" is carried in the IP protocol type field. The instruction set is restricted to pre-defined methods that correspond to the known protocol type field values and implement the standardized functionality specified by the IETF.

Active Storage

It would be advantageous if capsules could leave information behind in a node's non-transient storage. One might open a connection by arranging for a capsule to be executed at each node along a specific path, and having it leave a small amount of associated state in each node it traverses. Subsequent packets following this path would include code that locates and evaluates the connection state at each node.

A similar approach could be used to realize "flows" [9], which are somewhat softer than connections. Every capsule would include code that attempts to locate and use its "flow" state at the nodes it traverses. However, flow capsules are somewhat more robust than those used during connections in that the flow state is not essential to the capsule's successful execution. If a flow capsule encounters a node that has no relevant state information, it dynamically generates the required data, uses it for its own purposes, and leaves it behind for the convenience of later capsules. The network nodes can treat flow states as "soft state" values that are cached and can be disposed of if necessary. In this respect, flows are less demanding than connections on the robustness of node storage. Of course, our active connections and flows are considerably more powerful than those of present day systems, in that the "state" left behind is in the form of programs rather than static table entries.

Eventually, we hope to develop new schemes that go beyond traditional connections and flows. For example, capsules could be programmed to rendezvous at a node by arranging for the first arriving capsule to set some state information and then "sleep" until the remaining capsules have arrived. The capsules could then engage in some joint computation, such as may be used in sensor fusion applications or the pruning of multi-cast trees.

Finally, we note that capsules capable of modifying the node's storage provide a uniform mechanism for the implementation of background node functions. Routing protocols and table updates could be implemented in capsules as could network management functions, such as those provided by SNMP. Long-lived housekeeping functions could also be implemented in this manner, though in their case the "transient" execution environment might survive until the node is reset.

Program Extensibility

Unless programs are short relative to the data they encapsulate, it will prove inefficient for them to be carried in individual messages. Accordingly, it makes sense for the programming environment to be extensible, so that capsules can "plant" uniquely named classes and methods at nodes, for reference by other capsules and methods. In this way, most capsules can be concise – possibly a single instruction that invokes a user-specific method on the remainder of the capsule contents.

An interesting scheme would be to provide a mechanism that dynamically resolves references to external methods. Instead of capsules explicitly loading methods into the non-transient storage of the node, the node could contain a "cache" of known external methods and be equipped with a mechanism that allows it to locate and dynamically load methods on demand.[1] Although such a "demand" approach might suffer latency problems when a new application is started, this could be offset by allowing capsules that prime the cache when faults are anticipated.

The distinction between the "explicit" and "demand" loading schemes is closely related to the broader distinction we have made between discrete and integrated approaches to active networks. The explicit and discrete cases distinguish program loading as an explicit activity that must be completed prior to usage. In contrast, the demand and integrated cases offer increased flexibility with respect to determination and timing. Of course, this flexibility comes at some cost in terms of the sophistication of the mechanisms required to support safe and efficient loading.

3.4 Towards an Interoperable Programming Model

To be of general utility, capsules require: mobility, so that programs can be transmitted across the network; and portability, so that they can be loaded into a range of platforms. This suggests the development of a relatively small number of standardized models for the programming of network nodes and the description and allocation of their resources. Our objectives for such models are that they support:

- Mobility – the ability to transfer capsules and execute them on a range of platforms that leverage different underlying technologies.
- Safety – the ability to restrict the resources that capsules can access.
- Efficiency – enabling the above without compromising network performance, at least in the most common cases.

Traditional packet networks achieve interoperability by standardizing the syntax and semantics of packets. For example, Internet routers all support the agreed IP specifications; although router implementations may

---

[1] An interesting approach to resolving cache "faults" would be for a node to request the method from the node that sent it the faulting capsule – forming a chain back to the originator of the capsule who could be expected to take ultimate responsibility for resolving the reference. Of course, the originator might do so by demand loading the code from their software vendor.

differ, they implement roughly "equivalent" programs. In contrast, active nodes can execute many different programs, i.e., they can perform very different computations on the packets flowing through them. Network interoperability is achieved at a higher level of abstraction – instead of standardizing the computation performed on every packet, we standardize the computational model, i.e., the instruction set and resources available to capsule programs.

We find it convenient to distinguish between: issues surrounding the representation and evaluation of the capsules themselves; and safe access to node resources.

In section 4, we outline the functionality that is required and discuss mechanisms that can be used to support the safe and efficient execution of capsules. We discuss how "active technologies", developed within the programming language and operating system communities, can be used to prevent unauthorized access to resources that lie beyond the boundaries of the transient execution environment into which a capsule is dropped.

In section 5, we discuss resource management, which is an important issue in active nodes – these nodes are components of the shared infrastructure and their users must be protected from each other to a degree that is typically not required within personal computers or even group servers. The programming model must deal with two issues associated with node resources: interoperability and resource management. The interoperability requirement is for a common model of the resources available at a node. Resource management issues include resource allocation and capsule authentication and authorization.

4. CAPSULE PROGRAMS – MOBILITY, SAFETY AND EFFICIENCY

In this section we discuss: the languages in which capsule programs are expressed; and mechanisms that can support their safe and efficient execution. Our overall approach is to evaluate each capsule within the context of a transient execution environment whose lifetime is the interval during which the capsule is evaluated at a given node. Safety properties are provided by restricting the actions that can be performed and their scope, e.g., their access to storage and other node resources.

4.1 Capsule Primitives

There is a limited set of primitive actions that capsules can perform without straying beyond their transient environments. These actions constitute a restricted programming language, or instruction set, that can perform: arithmetic and branch operations; and manipulate stack and heap storage of the transient environment.

The set of primitive actions will be extended through the addition of external method invocation, which provides access to resources beyond the transient environment. Some of these external methods will leverage the same primitive actions and will also be evaluated in a closed environment, i.e., with a sharp distinction between their self-contained actions and their access to other methods. Others will access the built-in "API" of the node's run-time environment or embedded operating system. The API of active network nodes will be distinguished by the availability of methods that are tailored to the network environment, such as the efficient copying of capsules and sophisticated control over the scheduling of transmission resources.

For interoperability purposes all of the active nodes along a capsule's path should be capable of evaluating the capsule's contents.[2] There are three well known ways of achieving this level of portability/mobility:

- Express the programs in a high-level source language that may be interpreted at the nodes;
- Adopt a platform independent intermediate representation, typically a byte coded "virtual" instruction set;
- Express the programs in a platform-dependent binary format and arrange for each capsule to carry multiple encodings of its program – one for each type of platform that it traverses.

We expect to leverage all three approaches. Source encodings will prove useful for rapid prototyping. An intermediate representation may provide a compact and relatively efficient way to express relatively short programs. External methods could be similarly encoded or could be expressed in a machine dependent format. We expect that heavily used components will be packaged in binary libraries, especially "bootstrap" libraries that allow administrators to initialize newly installed or repaired platforms.

4.2 Safe and Efficient Execution

One of the reasons that we believe it will be possible to realize active networks is the availability of active technologies – mechanisms that allow users to inject customized programs into shared resources. Active technologies are not new. However, our use of active technologies within the network is novel – until now the use of active technologies in networking has been end-to-end (e.g., shipping code from servers to clients or vice-versa). In the case of active networks, the shared resources in question are the routers, switches and servers that lie within the network. Our

[2] At each hop, a capsule could translate itself into the language that is understood by the "next hop" node along its path. However, this approach seems extreme – even by our standards!

work will leverage and extend the presently available technologies.

Active Technologies – Background

Active technologies have been emerging in the fields of operating systems and programming languages for over ten years. Early work tended to address only one of three important issues – mobility, efficiency or safety. PostScript is an example of an early effort that stressed mobility over safety. Applications generate mobile programs that are executed at printers, which may be shared and distributed about a network.

In the field of parallel processing, "active messages" [10, 11] stressed efficiency over mobility by reducing the "program" to a single instruction – each message invokes an application-specific handler resident at the recipient. The handler provides a low overhead mechanism for dispatching arriving messages, so that they can be treated as self-scheduling computations. These systems, which targeted communication internal to a single parallel processor complex, did not address the safety issues relevant to shared infrastructures.

The advent of heterogeneous distributed systems and internetworking has accelerated the pace of research. The x-kernel [12] supports the composition of protocol handlers by providing a regular architecture for stacking them and by automating the dispatch process. Other efforts [13-15] have focused on less friendly environments by improving both the safety and efficiency with which handlers can be implemented. Most recently, there have been efforts to jointly address all three issues – mobility, safety and efficiency – under the banners of configurable operating systems, agents, mobile applets and other schemes related to the World Wide Web.

Leveraging Active Technologies

Active networks will adapt and extend active technologies for use within the network. In general, these technologies provide for safe execution by restricting the set of primitive actions available to mobile programs and the scope of their operands, e.g., their access to storage and other resources. An interesting question is how to organize the closures which provide the basis for safe execution. Our starting position is that the namespace of a capsule is restricted to the transient environment. This containment policy may be relaxed by initializing the closure with a default set of foundation components that all capsules are allowed to dispatch. Any capsule that accesses methods outside of that space must first request that its closure be extended by authenticating itself to a mechanism that validates its authorization.

In the following paragraphs, we discuss the available technologies in terms of the program encoding approaches – source, intermediate, or platform dependent binary – and then introduce "on-the-fly" compilation, a complementary technology that is also of interest.

Source Code

Safe-Tcl [16] is an example of a language that achieves safety through interpretation of a source program and closure of its namespace. The safety of such a system derives from the restricted closure and the correctness of the interpreter, which can prevent programs from deliberately or accidentally straying beyond the transient execution environment. The advantage of the Tcl-based approach is that programs are human readable and simple programs can be composed quite quickly. Furthermore, Tcl's character-based representation makes it easy to design programs that generate new source fragments. The principal disadvantage is the overhead of source code interpretation, which is compounded by Tcl's encoding of all data types as strings. An additional disadvantage is the overall size of programs, which could be reduced, albeit at the expense of readability, through the use of compression.

Intermediate Code

Java [17] achieves mobility through the use of an intermediate instruction set [18]. Traditionally, the safe execution of intermediate code has relied on the careful interpretation of the intermediate instruction set. One of Java's key contributions is the observation that a significant improvement in efficiency can be achieved by off-loading some of the responsibility from the interpreter. The instruction set, and its approved usage, are designed so as to reduce the degree of operand validation that the interpreter must perform as each instruction is executed. In part, this is enabled by the design of the instruction set, which precludes certain cases that would normally have to be checked. It is also enabled by the static inspection of code before it is first executed, so that many of the checks need only be performed once, typically when the program is first loaded.

Platform-dependent (Binary) Code

The most aggressive of the active technologies provide for the execution of platform-dependent binary programs that, for the most part, are directly executed by the underlying hardware. To safely execute such program fragments, one must restrict their use of the instruction set and address space. The traditional operating systems approach has been to rely on fairly heavyweight mechanisms, such as processes and hardware-supported address space protection. However, there has recently been progress on two lighter-weight approaches:

- The SPIN project [14] relies on the properties of the Modula 3 language and a trustworthy compiler to generate programs that will not stray beyond a restricted environment. When a program is presented for execution, the run-time system verifies that the instruction sequence was

generated by a trusted compiler and has not been modified.

- The approach described in [19, 20] prescribes a set of rules that instruction sequences must adhere to, such as restrictions on how address arithmetic is performed. In conjunction with a modicum of run-time support and a collection of clever techniques, these rules define a "sandbox" within which the program can do what it likes, but that it may not escape from. An important aspect of this work is that conformance to the "rules" can be statically verified when an instruction sequence is presented for execution.

In both cases, it is assumed that sophisticated compiler technology will be used to generate "safe" code. The distinction is whether the code is independently validated by the receiving platform or whether the compilers and/or vendors of programs are trusted and authorized to "sign for" their code.[3] The former approach improves mobility, especially across administrative boundaries. The latter approach not only saves the overhead of validation, but might also allow the compiler to generate code that is more efficient. In both cases, we would expect the directly executable binary code to out-perform an interpreted format.

<u>On-the-fly Compilation</u>

Recent work [21] has enabled "on-the-fly" compilation with a dialect of the C programming language. This allows source programs to be automatically tailored, or even wholly generated, at run-time. In conjunction with sandboxing, such a technology could allow active nodes to perform their own source-binary translations on capsules they are processing.

On-the-fly compilation technologies may prove crucial to the viability of our architecture. Modern IP routers achieve reasonable performance through careful tuning of their "fast paths", typically by optimizing a minimal instruction sequence that processes the vast majority of the traffic and relegates the more complex (and less frequently used) cases to other modules. An active node might achieve a similar performance boost by monitoring its traffic and dynamically generating a fast path program that streamlines the execution of the most common capsule programs. Techniques such as scheduling by path (found in Scout [22]) may also be applicable.

Discussion

Variability in network applications and traffic patterns suggests that there is no right answer. Although the performance that can be gained through binary encodings is attractive, it comes at the cost of portability. Furthermore, the instruction encodings associated with modern processors are far from compact – these schemes might give rise to much larger capsules than an intermediate encoding, suggesting a trade-off between transmission bandwidth and processing capacity. Finally, node implementors designing for high risk environments, i.e., focusing on safety, may prefer interpretation-driven schemes that audit the execution of each instruction.

Our plan is to adopt a Java-like instruction set as the basis for ActiveNet interoperability and code mobility. One of the benefits of the present IP packet format is that it enables an "hourglass" architecture in which a variety of upper layer protocols can operate over a wide range of network substrates. An intermediate instruction set will provide an analogous hourglass that facilitates mobility. A range of programming languages and compilers can be used to generate highly mobile intermediate code that can be executed on a wide range of hardware platforms.

Nonetheless, we believe that it will also prove practical and attractive to provide extensions that allow users and node implementors to leverage source and binary technologies. The architectural trick will be to enable these technologies, while retaining the intermediate instruction set as a fallback point that ensures interoperability. We have considered the following extensions:

- Allow programmers to optimistically leverage a source programming language, such as Safe-Tcl, in the hope that it is supported at the nodes a capsule traverses. A node that is not equipped with the appropriate interpreter or translator could either demand load one or forward the capsule to some other node that can translate it to the intermediate representation.
- Allow "fat" capsules that carry binary encodings (for popular platforms) alongside their intermediate encodings.
- Have nodes track their use of external methods, identifying candidates for binary encoding. A node could leverage on-the-fly technology to translate such methods locally, or it could load platform-specific versions from elsewhere on the network.
- The previous suggestion might be combined with demand loading. A node can identify its platform type whenever it requests an external method, affording the supplier the option of returning a binary encoding should an appropriate one be readily available.

4.3 Summary

In this section, we have outlined our approach to the safe and efficient evaluation of capsules. Although it is useful to distinguish between the program representation and its implementation, we realize that the two will strongly influence each other. The choice

[3] We assume the availability of appropriate authentication and tamper-proof signature technology.

of programming environment is going to preferentially favor certain implementation strategies, and at the same time implementation strategies that lead to efficiency or greater security (or simply become more popular) are going to influence the programming environment.

Having identified the requirement for common programming models, we are not suggesting that a single model be immediately standardized. The tensions between available programming models and implementation technologies can sort themselves out in the research "marketplace" as diverse experimental systems are developed, fielded, and accepted or rejected by users. For example, if the marketplace identifies two or three encodings as viable, then nodes that concurrently support all of them will emerge. As systems evolve to incorporate the best features of their competitors, we expect that a few schemes will become dominant.

5. NODE RESOURCES – INTEROPERABILITY AND SAFETY

Active networks will provide the building blocks for a shared information infrastructure that transcends many administrative domains. Accordingly, their design must address a range of "sharing" issues that are often brushed over in systems that are used in less public environments. We focus on two of the issues that must be addressed. For interoperability, capsule programmers must have a shared understanding as to what the resources are and how they are named. Secondly, mechanisms must be provided to limit access to scarce or sensitive resources.[4]

5.1 Interoperability – Resource Specification

The complexity of a system in which every capsule leverages a wide range of resources – each of which must be named, have its attributes specified and be carefully allocated – could explode quite quickly. Fortunately, most capsules will not require sophisticated resource models. We propose a relatively spartan approach employing a small set of platform independent abstractions for the physical resources of a node: transmission bandwidth, processing capacity, and transient storage. Additional flexibility is provided through longer term storage and logical resources, used by advanced applications, such as topology discovery, routing, and network management.

Transmission Bandwidth

Link bandwidth is typically not considered by the scheduling or resource allocation schemes of conventional operating systems. The link abstraction must encompass the units of bandwidth allocation and may take account of the traffic patterns that are generated. A detailed approach could draw on the service model [23] activities of the IETF. In some environments, simpler schemes may be possible, e.g., allowing each capsule program to consume a quantity of transmission bandwidth that is proportional to the size of the capsule it arrived in.

Instruction Execution (CPU)

It is somewhat easier to abstract a node's instruction processing resources – even multiprocessors tend to be homogeneous and their aggregate capacity is more or less established through industry benchmarks. In many cases, it will be sufficient to assign every capsule a default allocation that guards against runaway computations. However, the ability to trade computation against bandwidth may be useful to encourage, for example, compression prior to transmission on low bandwidth links.

Transient Storage

The transient execution environment consumes short term storage, which might also be limited. We tend to think of storage capacity along two axes: the storage utilized during specific intervals and the duration of those intervals. The former can be addressed by placing a default bound on the transient storage that can be allocated during capsule evaluation. The latter is somewhat trickier. We expect that most capsules will complete their execution quickly, i.e., in a few milliseconds or less. However, some capsules may linger, especially those that must rendezvous with others. This issue might be addressed by establishing a policy that permits the "garbage collection" of inactive capsules during times of shortage and requires capsules that are deliberately "sleeping" to place themselves in hibernation within longer term active storage.

Active Storage

We have identified requirements for the storage of components, such as external methods and data, that survive the execution of individual capsules. We find it useful to distinguish between two types of active storage, soft and persistent. Soft storage is used to cache objects, such as "hints", "flow state" or demand loaded components, that do not survive the re-initialization of a node. They can be deleted from the store without notice and their contents regenerated or reacquired if they are later needed. Given that this space is easily reclaimed, limits on its allocation may not be as important as the strategy that selects "victims" for reclamation[5]. Persistent storage provides a longer term abstraction for information that must be reliably stored, such as logs that are intended for

[4]There may be a further requirement to control the scheduling of some resources, such as transmission bandwidth. There may also be requirements for resource metering, accounting and/or auditing.

[5]Mechanisms such as those described in [24] might be used to "page" soft state to/from nearby nodes.

accounting and auditing purposes. This storage may also be used by applications that implement asynchronous multi-cast services, such as news distribution groups. Although this storage abstraction will be available at most nodes, it may be implemented by accessing replicated storage services located elsewhere on the network. We hope to leverage technologies such as [25] for this purpose.

Logical Resources

Although there are a relatively small number of physical resources, a node may support a large number of logical resources of many different types. This suggests the need for a uniform (not necessarily global) mechanism for naming instances of logical resources, including dynamically created resources, such as soft and persistently stored components. Fortunately, there is an abundance of past work on object naming schemes.

The class specifications of many logical resources, such as application-specific external methods or flow states, may be private in the sense that they need only be known to capsules generated by the relevant application. However, there will be a need for interoperable class specifications for some resources, such as routing tables. In this case, we hope to leverage existing notations, such as those used for SNMP Management Information Bases (MIBs). Where possible, we will leverage the existing MIB specifications themselves, which should facilitate interoperation between the ActiveNet and the existing Internet.

5.2 Resource Safety

The safe manipulation of node resources can be partitioned into three types of activities:

- dynamic, yet safe, assignment of resources to specific capsules.
- validation of user requests for resource assignment, through authentication and verification of their authorizations.
- automated delegation of resource authorizations.

Dynamic Assignment

Recall that our overall plan is to leverage the closure/addressability limitations enforced by active technologies. Resources are always represented and accessed through external methods and the default resources available to a capsule are included in the closure with which it is initiated. There is a further requirement for a mechanism that supports dynamic resource allocation. This can be accomplished by providing an external method that allows a program to request the safe "extension" of its closure.

Validation

The mechanism performing validation must authenticate the capsule source, check that it is authorized to access the resource and (possibly) verify that the resource request has not been tampered with. This mechanism need only be used in conjunction with requests for additional resources.

We assume that cryptography will provide the basis for the validation mechanism, but we may use a combination of schemes to reduce per-capsule overheads. For example, a public key scheme could be used to perform an initial authentication that establishes "soft state" that is then used by a lighter weight per-capsule signature algorithm. We are particularly interested in recent work on inexpensive techniques that provide less security for individual messages, but defend against large scale attacks [26].

Delegation

The preceding section assumed that the validation mechanism has access to information concerning authorizations, e.g., policy-initiated decisions as to the resources that can be made accessible to specific users or applications. We require a mechanism that supports the automated delegation of authorizations, in accordance with a straightforward model that both implementors and administrators can reason about. This issue was previously considered within the context of time-sharing systems [27, 28] but we are not aware of work that addresses delegation in as complex a system as the ActiveNet. Work on the cascading [29] and logic [30] of authentication, which has some of the delegation flavor we are looking for, may provide a starting point for further research.

Ultimately, this may be one of the most important "open" questions with respect to active networks. We envision an ActiveNet with as many or more administrative domains as the Internet (which is still growing), and administrators will be swamped if they are expected to manually coordinate the detailed authorization information This is another place where complexity, in this case administrative complexity, could overwhelm the infrastructure.

6. FROM INTERNET TO ACTIVENET

We suggest that interested researchers pool their talents in an effort to deploy a wide area ActiveNet. This experimental infrastructure could be overlaid on existing substrates, such as the Internet and the VBNS, obviating the need for dedicated transmission facilities. Although most of the ActiveNet nodes could be located at participating research sites, provision should be made to locate nodes at strategic locations not normally accessible to researchers, e.g., the NAPs of the Internet. If a research ActiveNet proves successful, it could be extended to assume direct control over the underlying transmission resources.

In assembling a collection of nodes into an ActiveNet it will be necessary to deal with many of the issues that have been addressed in the design of the current Internet – topology discovery, routing, etc. Initially, we expect to adopt the techniques used in the Internet. However, researchers should also investigate

new algorithms that leverage the availability of active nodes.

Eventually, it will be important to converge on an interoperable programming model that will achieve for active networks what IP standardization has for the Internet. However, the connectivity available through existing substrates will makes it possible to deploy a multiplicity of programming models in parallel, affording the research community an opportunity to explore alternative programming models and node implementations. It will be particularly important to engage application developers and users in the development of customized software components that exercise this "architecture of architectures".

7. ARCHITECTURAL CONSIDERATIONS

Conventional network architectures separate the upper (end-to-end) layers from the lower (hop-by-hop) layers. The network layer bridges these domains and enables interoperability by providing a fixed application- and user- neutral service that supports the exchange of opaque data between end systems.

Active networks challenge this traditional thinking in a number of ways: the computations performed within the network can be dynamically varied; they can be user- and application-specific; and the user data is accessible to them. We realize that this break with tradition raises a number of important questions, some of which are addressed in the following responsa.

How is interoperability achieved?

The key to interoperability is the network layer service, which is at the narrow point of the "hourglass". In the case of the Internet [9], there is a detailed specification of the syntax and semantics of the IP protocol, which must be implemented by all of the routers and communicating end systems. In effect, interoperability is supported by requiring that all of the nodes perform "equivalent" computations on the packets flowing through them.

In contrast, active nodes are capable of performing many different computations (i.e., executing many different programs) for different groups of users. However, the nodes must all support an "equivalent" computation model. Thus, network layer interoperability is based on an agreed program encoding and computation environment instead of a standardized packet format and fixed computation.

Architecturally, we are bumping up the level of abstraction at which interoperability is realized. There is still an hourglass – but the abstraction at its thinnest point has been made programmable.

Isn't the trend towards less functionality in the network?

The long term trend has actually been towards increased computation within the network. Whereas telephony circuit switches restrict computation to call setup time, packet switches perform computations on the header of every packet flowing through them. Active nodes extend the domain of computation to include the user data.

It is the "intelligence" or control over the network-based computation that has been migrating to the edges, allowing users to exercise greater control over their networks. Experience suggests that the two go hand in hand – increasing the flexibility of the computation performed within the network enables the deployment of even greater computational power at the edges.

What's the impact on the layered reference model?

There is presently a disconnect between what users consider to be "inside" the network and the practitioner's perspective, which is somewhat restricted. For example, web browsers allow users to interact with what they perceive to be "the network" without distinguishing among the many routers, domain name servers, and web servers that conspire to provide the service. It may be time for practitioners to re-evaluate their abstractions and start thinking about the network at a higher level.

Current thinking concerning network architecture has its roots in the layering of abstractions codified in the OSI Reference Model [31]. Although the model has proven quite useful it is showing cracks that should be addressed:

- Services at or below the network layer are presumed to be user- and application- neutral.
- It deals poorly with upper layer services that are physically interposed between communicating end points. Application relays can model these cases, but they are far from elegant.
- It does not model the "recursion" that occurs at the network layer, i.e., the tunneling of networks over each other.
- The upper layers, which have never been particularly satisfactory, are of diminished importance, given that active technologies enable the exchange of modules that implement application-specific protocols.

We are not certain what form a new model might take, but suggest that it will be more component-based than layered [32]. It might distinguish primitive functions, such as cell relaying and IP "fast paths", from computationally active functions, including those that configure the fast path components. Architecturally, these two types of components might be viewed as peers rather than layered upon each other. Such an architecture might also give rise to new hardware activities, such as the development of switching technology that "caches" fast paths and is highly responsive to active capsules.

What about the end-to-end argument?

The "end-to-end argument" [33] concerns the design of intermediaries, such as networks, that provide

services that cannot be made perfectly reliable. Since users of these services must provide "end-to-end" mechanisms that cope with failures, designers are counseled against over-engineering the intermediaries by adding significantly to their complexity or overhead, i.e., by trying to make them "almost" perfectly reliable. Instead, the designer's objective should be an "acceptable" level of reliability that does not trigger excessive intervention by the end-to-end mechanism. The designer is encouraged to strike a balance by relying on end-to-end mechanisms to ensure correctness, and by leveraging simple and optimistic mechanisms to enhance performance, where appropriate.

While locating computation within the network may appear to contradict this guideline, we note that the argument pertains to the placement of functionality – it does not suggest that the choice of functions that are appropriately located within the network cannot be application-specific. If anything, active networks allow this guideline to be followed more carefully, by allowing applications to selectively determine the partitioning of functionality between the end points and intermediaries.

Why hasn't this been done before? Why try now?

The approach we are proposing synthesizes a number of technologies: programmable node platforms, component-based software engineering, and code mobility. A few "programmable" networks have been developed in the past, and suggestions for object-based approaches to network implementation surface every few years. However, the previous work has not leveraged code mobility within the network, let alone within the context of each and every capsule or packet.

A key enabler of our approach is the availability of "active technologies" that enable safe and efficient code mobility. The absence of these technologies would have precluded similar projects in the past – and their recent emergence underscores the timeliness of the proposed effort.

8. CONCLUSIONS

In this paper we have described our vision of an active network architecture that can be programmed by its users. We have also called for community participation in an effort to develop and deploy a research ActiveNet. In the course of this presentation we have raised a number of architectural issues and research questions that remain to be addressed.

We expect that active networks will enable a range of new applications in addition to the lead applications that already rely on the interposition of customized computation within the network. However, we believe that this work will also have broader implications, on how we think about networks and their protocols; and on the infrastructure innovation process.

Programming the Network

We are applying a programming language perspective to networks and their protocols. In place of protocol "stacks", we anticipate the development of protocol components that can be tailored and composed to perform application-specific functions. These protocol components will leverage the tools of the modern programming trade – encapsulation, polymorphism and inheritance. Within our own research group, we are setting out to create a "Smalltalk of networking" and are interested not just in the "language" itself but also in the class hierarchy, etc. that will develop around it.

Our enthusiasm is tempered by the realization that suggestions for object-oriented approaches to networking surface every five to ten years, and have had little impact on mainstream research. However, we believe that it is now time to make a large scale effort towards their realization. The availability of active technologies and lead applications – in conjunction with rising processing power and bandwidth – presents opportunities that were not previously available.

Infrastructure Innovation

As the Internet grows it is increasingly difficult to maintain, let alone accelerate, the pace of innovation. Today, after a concept is prototyped its large scale deployment takes about 8 years. The process involves standardization, incorporation into vendor hardware platforms, user procurement and installation. The present backlog within the IETF includes multicast, authentication and mobility extensions, RSVP and IPv6.

Active networks will address the mismatch between the rate at which user requirements can change, i.e., overnight, and the pace at which physical assets can be deployed. They will accelerate the pace of innovation by decoupling network services from the underlying hardware and by allowing new services to be demand loaded into the infrastructure. In the same way that IP enabled a range of upper layer protocols and transmission substrates, active networks will facilitate the development of new network services and hardware platforms.

Conventional network routers are based on proprietary hardware platforms that are bundled with customized software. Active networks present an opportunity to change the structure of the networking industry, from a "mainframe" mind-set, in which hardware and software are bundled together, to a "virtualized" approach in which hardware and software innovation are decoupled [34]. A market for "shrink-wrapped" network software will facilitate innovation by:

- Allowing third parties to develop innovative software without customizing their products to a specific platform.

- Removing the software barrier to entry that discourages new players from fielding innovative hardware.
- Addressing the "chicken and egg" problem associated with new services – vendors are hesitant to support services before they gain user acceptance, yet the utility of many network services is dependent on their widespread availability.

Furthermore, the process will be user-driven. The widespread availability of new services will depend on their acceptance in the marketplace, without being delayed by vendor consensus and standardization activities. Similarly, hardware vendors will seek competitive advantage by optimizing their platforms to suit changing workloads.

Summary

Active networks appear to break many of the architectural rules that conventional wisdom holds inviolate. However, we believe that they build on past successes with packet approaches, such as the Internet, and at the same time relax a number of architectural limitations that may now be artifacts of previous generations of hardware and software technology.

Passive network architectures that emphasize packet-based end-to-end communication have served us well. However, as our lead users demonstrate, computation within the network is already happening – and it would be unfortunate if network architects were the last to notice. It is now time to explore new architectural models, such as active networks, that incorporate interposed computation. We believe that such efforts will enable new generations of networks that are highly flexible and accelerate the pace of infrastructure innovation.

ACKNOWLEDGMENTS

The authors wish to thank: Jennifer Steiner Klein, who assisted in the drafting and technical editing of the manuscript; Rachel Bredemeier who assisted with layout and the bibliography; and Sun Microsystems Laboratories, who provided "seed" funding for this project. This work has been influenced by discussions with a number of researchers, especially: Deborah Estrin, Henry Fuchs, Butler Lampson, Paul Leach, Gary Minden, Herb Schorr, Scott Shenker and Dave Sincoskie.

REFERENCES

1. Chankhuntod, A., P.B. Danzig, and C. Neerdaels. *A Hierarchical Internet Object Cache*. in *Proceedings of 1996 USENIX*. 1996.
2. Blaze, M. and R. Alonso. *Dynamic Hierarchical Caching in Large-Scale Distributed File Systems*. in *12th Intl. Conf. on Distributed Computing Systems*. 1992. Yokohama, Japan.
3. Gwertzman, J.S. and M. Seltzer. *The Case for Geographical Push-Caching*. in *1995 Workshop on Hot Operating Systems*. 1995.
4. Kleinrock, L. *Nomadic Computing (Keynote Address)*. in *Int. Conf. on Mobile Computing and Networking*. 1995. Berkeley, CA.
5. Balakrishnan, H., *et al. Improving TCP/IP Performance over Wireless Networks*. in *Int. Conf. on Mobile Computing and Networking*. 1995. Berkeley, CA.
6. Amir, E., S. McCanne, and H. Zhang. *An Application Level Video Gateway*. in *ACM Multimedia '95*. 1995. San Francisco, CA.
7. Le, M.T., F. Burghardt, and J. Rabaey. *Software Architecture of the Infopad System*. in *Mobidata Workshop on Mobile and Wireless Information Systems*. 1994. New Brunswick, NJ.
8. Pasquale, J.C., *et al. Filter Propagation in Dissemination Trees: Trading Off Bandwidth and Processing in Continuous Media Networks*. in *4th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*. 1993.
9. Clark, D.D. *The Design Philosophy of the DARPA Internet Protocols*. in *ACM Sigcomm Symposium*. 1988. Stanford, CA.
10. von Eicken, T., *et al. Active Messages: A Mechanism for Integrated Communication and Computation*. in *19th Int. Symp. on Computer Architecture*. 1992. Gold Coast, Australia.
11. Agarwal, A., *et al. The MIT Alewife Machine: Architecture and Performance*. in *22nd Int. Symp. on Computer Architecture (ISCA '95)*. 1995.
12. O'Malley, S.W. and L.L. Peterson, *A Dynamic Network Architecture*. ACM Transactions on Computer Systems, 1992. 10(2) p. 110-143.
13. Jones, M. *Interposition Agents: Transparently Interposing User Code at the System Interface*. in *14th ACM Symp. on Operating Systems Principles*. 1993. Asheville, NC.
14. Bershad, B., *et al. Extensibility, Safety and Performance in the SPIN Operating System*. in *15th ACM Symp. on Operating Systems Principles*. 1995.
15. Engler, D.R., M.F. Kaashoek, and J. O'Toole Jr. *Exokernel: An Operating System Architecture for Application-Level Resource Management*. in *15th ACM Symp. on Operating Systems Principles*. 1995.
16. Borenstein, N. *Email with a Mind of its Own: The Safe-Tcl Language for Enabled Mail*. in *IFIP International Conference*. 1994. Barcelona, Spain.
17. Gosling, J. and H. McGilton, *The Java Language Environment: A White Paper*, 1995, Sun Microsystems.
18. Gosling, J. *Java Intermediate Bytecodes*. in *SIGPLAN Workshop on Intermediate Representations (IR95)*. 1995. San Francisco, CA.

19. Wahbe, R., *et al. Efficient Software-Based Fault Isolation.* in *14th ACM Symp. on Operating Systems Principles.* 1993. Asheville, NC.

20. Colusa Software, *Omniware: A Universal Substrate for Mobile Code,* 1995, Colusa Software.

21. Engler, D.R., W.C. Hsieh, and M.F. Kaashoek. *'C: A Language for High-Level, Efficient, and Machine-Independent Dynamic Code Generation.* in *23rd Annual ACM Symp. on Principles of Programming Languages (to appear).* 1996. St. Petersburg, FL.

22. Montz, A.B., *et al., Scout: A Communications-Oriented Operating System,* 1994, Dept. of Computer Science, University of Arizona.

23. Shenker, S., D.D. Clark, and L. Zhang. *Services or Infrastructure: Why we Need a Network Service Model.* in *1st Int'l Workshop on Community Networking.* 1994. San Francisco, CA.

24. Anderson, T.E., *et al., A Case for Networks of Workstations: NOW.* IEEE Micro, 1995. 15(1) p. 54-64.

25. Liskov, B., *et al. Safe and Efficient Sharing of Persistent Objects in Thor.* in *SIGMOD '96.* 1996. Montreal, Canada.

26. Manasse, M.S. *The Millicent Protocols for Electronic Commerce.* in *1st USENIX Workshop on Electronic Commerce.* 1995. New York, NY.

27. Bell, D.E. and L.J. LaPadula, *Secure Computer Systems: Unified Exposition and Multics Interpretation,* 1976, MITRE Corp.

28. Wilkes, M.V. and R.M. Needham, *The Cambridge CAP Computer and Its Operating System.* Operating and Programming Systems Series, ed. P.J. Denning. 1979, New York City: North Holland.

29. Sollins, K.R. *Cascaded Authentication.* in *Proceedings of the 1988 IEEE Symposium on Security and Privacy.* 1988. Oakland, CA.

30. Burrows, M., M. Abadi, and R. Needham, *A Logic of Authentication.* ACM Transactions on Computer Systems, 1990. 8(1) p. 18-36.

31. ISO, *Information Processing Systems - Open Systems Interconnection - Basic Reference Model,* 1984, ISO.

32. Clark, D.D. and D.L. Tennenhouse. *Architectural Considerations for a New Generation of Protocols.* in *ACM Sigcomm Symposium.* 1990.

33. Saltzer, J.H., D.P. Reed, and D.D. Clark, *End-To-End Arguments in System Design.* ACM Transactions on Computer Systems, 1984. 2(4) p. 277-288.

34. Tennenhouse, D., *et al., Virtual Infrastructure: Putting Information Infrastructure on the Technology Curve.* Computer Networks and ISDN Systems (to appear), 1996.

ABSTRACT

Active networks are a novel approach to network architecture in which the switches of the network perform customized computations on the messages flowing through them. This approach is motivated by both lead user applications, which perform user-driven computation at nodes within the network today, and the emergence of mobile code technologies that make dynamic network service innovation attainable. In this article, the authors discuss two approaches to the realization of active networks and provide a snapshot of the current research issues and activities.

A Survey of Active Network Research

David L. Tennenhouse, Massachusetts Institute of Technology
Jonathan M. Smith, University of Pennsylvania
W. David Sincoskie, Bell Communications Research
David J. Wetherall, Massachusetts Institute of Technology
Gary J. Minden, University of Kansas

In an active network, the routers or switches of the network perform customized computations on the messages flowing through them. For example, a user of an active network could send a "trace" program to each router and arrange for the program to be executed when their packets are processed. Figure 1 illustrates how the routers of an IP network could be augmented to perform such customized processing on the datagrams flowing through them. These active routers could also interoperate with legacy routers, which transparently forward datagrams in the traditional manner.

These networks are active in the sense that nodes can perform computations on, and modify, the packet contents. In addition, this processing can be customized on a per-user or per-application basis. In contrast, the role of computation within traditional packet networks, such as the Internet, is extremely limited. Although routers may modify a packet's header, they pass the user data opaquely without examination or modification. Furthermore, the header computation and associated router actions are specified independent of the user process or application that generates the packet.

The concept of active networking emerged from discussions within the broad Defense Advanced Research Projects Agency (DARPA) research community in 1994 and 1995 on the future directions of networking systems. Several problems with today's networks were identified: the difficulty of integrating new technologies and standards into the shared network infrastructure, poor performance due to redundant operations at several protocol layers, and difficulty accommodating new services in the existing architectural model. Several strategies, collectively referred to as *active networking*, emerged to address these issues. The idea of messages carrying procedures and data is a natural step beyond traditional circuit and packet switching, and can be used to rapidly adapt the network to changing requirements. Coupled with a well understood execution environment within network nodes, this program-based approach provides a foundation for expressing networking systems as the composition of many smaller components with specific properties: services can be distributed and configured to meet the needs of applications, and statements can be made about overall network behavior in terms of the properties of individual components.

In this article we discuss two approaches to the realization of active networks. The *programmable switch* approach maintains the existing packet/cell format, and provides a discrete mechanism that supports the downloading of programs. Separating the injection of programs from the processing of messages may be particularly attractive when the selection of programs is made by network administrators rather than individual end users. In contrast, the *capsule* approach goes somewhat further — the passive packets of present-day architectures are replaced by active miniature programs that are encapsulated in transmission frames and executed at each node along their path. User data can be embedded within these *capsules*, in much the same way a page's contents are embedded within a fragment of PostScript code.

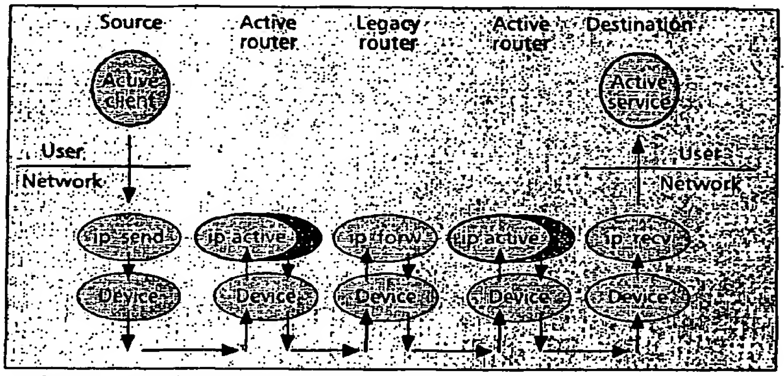
Research in active networks is motivated by both technology "push" and user "pull." The "pull" comes from the assortment of firewalls, web proxies, multicast routers, mobile proxies, video gateways, and so forth that perform user-driven computation at nodes "within" the network. Some of these *lead users* are described in Table 1. In many cases, these services are implemented at nodes, such as firewalls, which adopt the facade of routers but perform application-specific processing that transcends conventional architectural guidelines. Our goal is to replace the numerous ad hoc approaches to network-based computation with a generic capability that allows users to program their networks.

The technology "push" is the emergence of *active* technologies that make our goals attainable. Until recently, the specter of administrators (let alone end users) programming their networks has raised insurmountable concerns with

 0163-6804/97/$10.00 © 1997 IEEE

respect to infrastructure safety and efficiency. However, recent advances in programming languages, compilers, and operating systems may provide the keys to the safe and efficient execution of mobile program fragments. Today, these active technologies are applied within individual end systems and above the end-to-end network layer (e.g., to allow Web servers and clients to exchange Java applets). Active networks leverage and extend these technologies for use *within* the network, in ways that will fundamentally change our mindset concerning what is "in" the network.

This article provides a current snapshot of active network research activities, including work on the underlying active technologies. In the next two sections, we describe the impact active networks may have on infrastructure innovation and the new applications that will be enabled. We then present a framework, or set of issues, that can be used to categorize and organize activity within the field. Finally, we present a survey of current research activities within our own laboratories and elsewhere in the community.



■ **Figure 1.** *Application-specific processing within the nodes of an active network.*

ACCELERATING INFRASTRUCTURE INNOVATION

As the lead users cited in Table 1 demonstrate, computation within the network is already happening; the demonstrated demand for these services suggests that network architectures must adapt to deal with this new reality.

At a more fundamental level, the network innovation process is itself ripe for renewal. The pace of network innovation is far too slow; and as the Internet grows it is increasingly difficult to maintain, let alone accelerate, this pace. To a large degree this is a function of the need to achieve consensus: a network's utility increases with the number of interconnected nodes. Today, the path from prototype demonstration to large-scale deployment takes about ten years. The process involves standardization, incorporation into vendor hardware platforms, user procurement, and installation. The present backlog of Internet services includes multicast, authentication and mobility extensions, Reservation Protocol (RSVP), and Internet Protocol version 6 (IPv6).

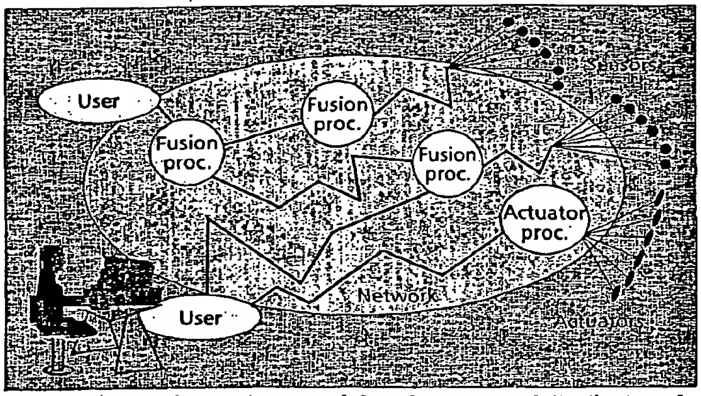
IP enables interoperability by defining a standard packet format and addressing scheme; although router implementations may differ, they implement roughly "equivalent" programs. Thus, the mechanisms for IP innovation are changing the IP service, which means changing everything (since it is the basis for interoperability); or establishing overlays (e.g., the MBone).

In contrast, active nodes can execute many different programs; that is, they can perform very different computations on each of the packets flowing through them. Instead of insisting that all the routers perform "equivalent" computations on every packet, active networks specify that all nodes support equivalent computational models (i.e., virtual instruction sets). Active networks raise the level of abstraction at which interoperability is realized, allowing applications to customize message processing to suit their purposes.

The ability to download new services into the infrastructure will lead to a user-driven innovation process in

Category	Description
Firewalls	Firewalls implement filters that determine which packets should be passed transparently and which should be blocked. Although they have a peer relationship to other routers, they implement application- and user-specific functions in addition to packet routing [1]. The need to update the firewall to enable the use of new protocols is an impediment to their adoption. In an active network, this process could be automated by allowing applications from approved vendors to authenticate themselves to the firewall and inject the appropriate modules into it.
Web proxies	Web proxies provide a user-transparent service tailored to the serving and caching of Web pages. Harvest [2] employs a hierarchical scheme in which cache nodes are located near the edges of the network, i.e., within end user organizations. This system could be extended by allowing nodes of the hierarchy to be located at strategic points within the network.
Nomadic routers	Kleinrock describes a "nomadic router" [3] interposed between an end system and the network. This module observes and adapts to the means by which the end system is connected to the network, e.g., through a phone line in a hotel room versus through the LAN in the home office. It might decide to perform more file caching or link compression when the end system is connected through a low bandwidth link and invoke additional caching security, such as encryption, when operating away from the home office.
Transport gateways	"Transport gateways" are nodes located at strategic points that bridge networks with vastly different bandwidth and reliability characteristics (e.g., at the junctions between wired and wireless networks). To support mobile access to wired networks, TCP snooping [4] retains per-connection state information at wireless base stations.
Application services	Application-specific gateways support services such as the transcoding of images [5] among video conference users with differing bandwidth constraints. Similarly, InfoPad [6] instantiates user-specific "pad servers," supporting applications such as voice and handwriting recognition, at intermediate nodes.

■ **Table 1.** *Lead users.*



■ Figure 2. *Exploiting the network-based merging and distribution of information. (Diagram courtesy of Prof. Henry Fuchs, UNC)*

which the availability of new services will be dependent on their acceptance in the marketplace. Active networks present an opportunity to change the structure of the networking industry from a "mainframe" mindset, in which hardware and software are bundled together, to a "virtualized" approach, in which hardware and software innovation are decoupled [7]. The network programming abstraction provides a powerful platform for user-driven customization of the infrastructure, allowing new services to be deployed at a faster pace than can be sustained by vendor-driven consensus and standardization activities.

Enabling New Applications

Active networks will enable new applications that rely on network-based merging of information, user-aware network protection, and active network management.

Merging and Distribution of Information

The era of multi-user, multisite applications has just begun; the success of the MBone and the Web are but harbingers of what may lie ahead. There is an untapped reservoir of applications that require network-based services to support the merging and distribution of information. However, existing systems are based on a service that provides an extremely limited function (i.e., the copying of IP packets) without support for application-specific distribution, let alone network-based storage or information fusion.

Figure 2 illustrates how sophisticated multisite applications will leverage computation and storage within the network. In this figure an application, such as simulation or remote manipulation, allows each user to "see" composite images constructed by fusing information obtained from a large number of sensors. Furthermore, each sensor can be "watched" by a number of users, who will have different requirements concerning the encoding and presentation of the information they access. Merging data within the network reduces the bandwidth requirements at the users, who are located at the (low-bandwidth) periphery of the network. Similarly, user-specific multicast services within the network reduce the load on the sensors and network backbone.

Web proxies that cache pages of information are another example of a multi-user service which could benefit from network-based computation and storage. Harvest [2] employs a hierarchical caching scheme that can reduce the latencies experienced by individual users and the aggregate bandwidth consumed. The cache nodes are presently located near the edges of the network, that is, at nodes within the end-user organizations. These systems could be extended by allowing nodes of the hierarchy to be located at strategic points within the networks of Internet access providers and interexchange carriers. An interesting problem is the development of algorithms and tools that automatically "balance" the hierarchy by repositioning the caches themselves, not just the cached information. A further argument in favor of using active technologies for Web caching is that a significant fraction of web pages are dynamically computed and not susceptible to passive caching. This suggests the development of schemes that support active caches which store and execute programs that generate these pages.

User-Aware Network Protection

Protection of information means that the right information gets to the right people at the right place and time. Although network security and authentication mechanisms are being proposed in many networking forums, active networking may admit the design of an integrated mechanism that governs all network resources and the information flowing through them. This eliminates the need for multiple security/authentication systems operating independently at each communication protocol layer. It allows us to program in security policy for the network on a per-user or per-use basis. Finally, a formal approach using rigorous specifications and language-enforced-type safety can be used to reason about the protection policies and the mechanisms of their implementation.

Active Network Management

Many network management tasks consist of collecting and collating data, such as event counts. To provide the most useful network management data, such as exception indications, intelligence must be used to filter out uninteresting (unexceptional) events. Active technologies could be used to implement sophisticated approaches to network monitoring and event filtering. Network components such as routers may even assume a degree of responsibility for monitoring themselves (e.g., by injecting customized monitoring and diagnostic programs into their nearest neighbors). Similarly, active networks can provide the flexibility necessary to improve fault detection and to update the survivability policies that govern component response to correlated failures, such as those caused by earthquakes or malicious intruders.

A Framework for Active Network Research

In this section, we distinguish two approaches to active networks, discrete and integrated, depending on whether programs and data are carried discretely (i.e., within separate messages) or in an integrated fashion. We then discuss common issues related to node programming and interoperability.

Programmable Switches — A Discrete Approach

The processing of messages may be architecturally separated from the business of injecting programs into the node, with a separate mechanism for each function. This preserves the current distinction between in-band data transfer and out-of-band management channels. Users would first inject their custom processing routines into the required routers. Then they would send their packets through such "programmable" nodes much as they do today. When a packet arrives at a node its header is examined, and the appropriate program is dispatched to operate on its contents.

Separate mechanisms for loading and execution might be

Project	M	S	E	Description
Safe Tcl [8] (source)	X	X		Safe Tcl (based on Tcl) is a scripting language that provides safety through interpretation of a source program and closure of its namespace. It depends on the restricted closure and correctness of the interpreter to prevent programs from deliberately or accidentally straying beyond their permitted execution environment.
Java [9] (intermediate)	X	X	X	Java uses an intermediate instruction set to achieve mobility. Traditionally, the safe execution of intermediate code has relied on its careful interpretation. One of Java's key contributions is to improve efficiency by offloading responsibility from the interpreter: the instruction set and its approved usage are designed to reduce operand validation per executed instruction. Work at the University of Arizona and elsewhere seeks to further boost efficiency through the use of compilation techniques.
Omniware [10] (object code)	X	X	X	Omniware portable object code depends on software-based fault isolation (SFI) to enforce safety efficiently. It prescribes a set of rules to which instruction sequences must adhere (e.g., restrictions on how address arithmetic is performed). In conjunction with runtime support, these rules define a "sandbox" within which the program can do what it likes, but from which it may not escape.
Proof-Carrying Code [11] (object code)		X	X	PCC uses a novel approach to achieve safety. It attaches a formal proof of the properties of a binary program. The recipient can check that the proof is valid, a much simpler process than constructing it from scratch. Currently, PCC is practical only for short programs.

■ **Table 2.** *Program encoding technolgoies (with labeled columns M, S, and E assessing mobility, safety, and efficiency, respectively).*

valuable when program loading must be carefully controlled. Allowing operators to dynamically load code into their routers would be useful for router extensibility purposes, even if the programs do not perform application- or user-specific computations. In the Internet, for example, program loading could be restricted to a router's operator who is furnished with a "back door" through which they can dynamically load code. This back door would at minimum authenticate the operator and might also perform extensive checks on the code being loaded.

CAPSULES — AN INTEGRATED APPROACH

A more extreme view of active networks is one in which *every* message is a program. Every message, or capsule, that passes between nodes contains a program fragment (of at least one instruction) which may include embedded data. When a capsule arrives at an active node, its contents are evaluated, in much the same way that a PostScript printer interprets the contents of each file sent to it.

Bits arriving on incoming links are processed by a mechanism that identifies capsule boundaries, possibly using the framing mechanisms provided by traditional link-layer protocols. The capsule's contents are then dispatched to a transient execution environment where they can safely be evaluated. We hypothesize that programs are composed of instructions, which perform basic computations on the capsule contents, and can also invoke "built-in" primitives, which may provide access to resources external to the transient environment. The execution of a capsule results in the scheduling of zero or more capsules for transmission on the outgoing links and may change the nontransient state of the node.

TOWARD A COMMON PROGRAMMING MODEL

Network programs must be transmitted across the communication substrate and loaded into a range of platforms. This suggests the development of common models for: the encoding of network programs; the "built-in" primitives available at each node; and the description and allocation of node resources.

Program Encoding — Our objectives for program encodings are that they support:
- *Mobility* — the ability to transfer programs and execute them on a range of platforms
- *Safety* — the ability to restrict the resources that programs can access
- *Efficiency* — enabling the above without compromising network performance, at least in the most common cases

Mobility may be achieved at several different levels of program representation: express the program in a high-level scripting language such as Tcl; adopt a platform-indepen-

Project	Description
Scout [13]	Scout is designed to support communication-oriented tasks. It allocates and schedules resources on a "path" basis and applies a number of optimizations intended to increase throughput and decrease latency. Many of the techniques may be applicable to programs loaded into network nodes.
Exokernel [14]	The exokernel enables programs to safely share low-level access to system resources. It implements a thin veneer that securely multiplexes the raw hardware. This in turn allows programs to tailor their own abstractions of operating system services (e.g., access to the active node environment).
SPIN [15]	SPIN relies on the properties of the Modula-3 language and a trustworthy compiler to generate programs that will not stray beyond a restricted environment. Programs signed by the compiler may be dynamically loaded into the operating system.
'C [16]	'C and VCODE enable "on-the-fly" code generation. This allows source programs to be automatically tailored, or even wholly generated, at runtime. These technologies could allow active nodes to translate commonly used programs to binary encodings.

■ **Table 3.** *Operating system technologies.*

dent intermediate representation, typically a byte-coded virtual instruction set (e.g., Java); or transfer programs in binary formats, such as Omniware. Table 2 describes recently developed enabling technologies that support the safe and efficient execution of each level of program encoding. We expect that all three approaches will prove useful: source encodings support rapid prototyping; intermediate representations provide a compact and relatively efficient way to express short programs; and commonly used modules might best be expressed at the object code level.

Connections and flows in active networks can be more powerful than those of present-day systems because the state left behind may be in the form of programs.

A possible approach to node interoperability would be to agree on an intermediate instruction encoding as the backstop for code mobility. Node implementors and users would be welcome to leverage alternative encodings, as long as they provide mechanisms through which an intermediate encoding of a program can be obtained or generated. Implementors may also leverage techniques, such as dynamic ("on-the-fly") compilation, that optimize common processing routines by both converting portable representations to native ones and specializing programs to individual contexts. Operating system support for more specific strategies, such as "path"-based scheduling, protocol code reorganization, and low-level extensibility should also prove useful. Table 3 describes some of these compilation and operating system technologies.

Common Primitives — The services built into each node might include several categories of operations [12]: primitives that allow the packet itself to be manipulated (e.g., by changing its header, payload, and/or length); primitives that provide access to the node's environment (e.g., node address, time of day, and link status); and primitives for controlling packet flow (e.g., forwarding, copying, and discarding). Additional primitives might provide access to node storage and scheduling, for example, to facilitate rendezvous operations that combine processing across multiple packets.

Node Resources and Their Allocation — Beyond encodings and primitives, there must be a common model of node resources and the means by which policies governing their allocation are communicated. The resources to be modeled include physical resources, such as transmission bandwidth, processing capacity, and storage, as well as logical resources, such as routing tables and the node's management information base. Safe resource allocation is an area that will require considerable attention. Active nodes will be embedded within the shared network infrastructure, so their designs must address a range of sharing issues that are often brushed aside in the design of programmable systems destined for less public environments.

Current Research

Work on active networks is underway at a number of sites which are independently studying: capsule and programmable switch architectures; enabling technologies; specification techniques; end system issues; and applications, including network, mobility, and congestion management.

Massachusetts Institute of Technology

The MIT team is prototyping an architecture based on the capsule approach [17] and studying issues related to component specification, "active storage," multicast negative acknowledgment (NACK) fusion, and network-based traffic filtering. A reference platform that demonstrates the capsule architecture is being implemented on Linux using Java-based capsule encoding. Additional enabling technologies, including advanced operating system techniques [14] and "on-the-fly" compilation [16], are also under investigation.

Capsules use the built-in constructs of a programming language to perform packet processing. This language will be extended through the specification of a suite of "foundation components" that invoke built-in primitives and interact with the local node environment, and can be extended and specialized to suit application-specific requirements. Demand loading and the caching of components are being developed as strategies to support compact programs and reduce the overhead associated with their transfer and evaluation. Demand loading allows capsules to reference components rather than carry them; and caching implies that recently used components need not be reloaded and verified for safety.

Programming will also be facilitated by allowing capsules to leave "soft state" behind in a node. Thus, a flow or connection may be opened by having a capsule leave a small amount of associated state at each node along the path it traverses. Subsequent packets can include code whose execution leverages this "soft state" but can regenerate it if necessary. Connections and flows in active networks can be more powerful than those of present-day systems because the state left behind may be in the form of programs. A more persistent form of active storage, workflow state, is being developed to support loosely synchronized activities and track dependencies.

University of Pennsylvania

The SwitchWare project [18] is developing a programmable switch approach that allows digitally signed type-checked modules to be loaded into the nodes of a network. The basic idea is to raise the level of abstraction of the switch functionality closer to that of a Turing machine. Aspects of security dictate limitations in the trade-offs which can be made in support of other goals: resource allocation must be robust enough that denial of service attacks are frustrated; extensibility must be restricted to preclude security breaches, yet still adequate for advanced applications.

Penn's approach uses formal methodologies to prove security properties of SwitchWare programs. The focus of SwitchWare is the identification of properties of the underlying infrastructure for which theorems can be developed. Proofs are supported by a language (SML/NJ) with a precise definition and runtime support that includes concurrent garbage collection and resource allocation. An advantage of supporting security at the programming language level is that the high overhead of protection domain-crossing in kernelized operating systems is avoided, since the need for carefully gated entry points is removed at compilation time.

The approach will be evaluated with a prototype based on a shared-memory multiprocessor. Early prototype applications include software-scalable bandwidth based on a general mechanism for inverse multiplexing (i.e., network striping) and support for an active packet model ("Switchlets").

Bell Communications Research

Several aspects of the Penn design will be studied jointly with Bellcore, using a different infrastructure (OPCV2) to extend the design space explored. The output port controller version

2 (OPCV2) attaches to the Sunshine asynchronous transfer mode (ATM) switch, developed for the AURORA gigabit testbed, and can also be used as a standalone processor that enables line speed manipulation of ATM streams. This allows studies of SwitchWare multiplexing algorithms and run-time system functionality to be embedded in the port controllers of a scalable switch.

We anticipate changes to the organization of end system software — in place of protocol "stacks," applications may use protocol "components" that can be specialized and composed to perform application-specific functions.

A second component of the Bellcore effort is the specification of the semantics of an active router, and the investigation of those semantics in a collaborative prototyping effort involving Penn. The prototype will use a small-scale multiprocessor as an active network element that interconnects ATM networks with 10 and 100 Mb/s Ethernets. This active router will serve as an experimental platform for the investigation of applications under development within the SwitchWare project.

Bellcore is also studying uses of the new network infrastructure, such as self-paying information transport, in which electronic payment information is embedded in the active packets. Bellcore's interest in active networks is related to its previous work on protocol boosters [19], which dynamically optimize protocol components on an end-to-end basis, and the advanced intelligent network (AIN), which separated the implementation of value added services from switching, by moving the service control functions to adjunct processors.

Columbia University

The NetScript project, led by Yemini and da Silva [20], consists of a programming language and execution environment. The language provides a means to script the processing of packet streams. It is particularly suited to the implementation of routing, packet analysis, signaling, and management functions. NetScript agents can be sent to remote systems including intermediate network nodes, such as routers and switches. The goal is to enable programming of these nodes as easily and quickly as end systems.

Carnegie Mellon University

The CMU team, led by Steenkiste and Zhang, is developing resource management mechanisms in support of "application-aware" networks. They are considering three dimensions of resource allocation: physical infrastructure, including processing and storage; decision making on different time scales, ranging from application startup to packet and cell scheduling; and the sharing of infrastructure among organizational entities. The mechanisms will support network customization across all three dimensions.

CMU is also exploring support for sophisticated multiparty applications, such as video conferencing and data mining, that use multiple traffic streams with divergent characteristics. These applications will be "network-aware" so that they can perform well on a variety of networks and adapt quickly to changing network conditions.

Work Elsewhere

Additional research on active networks is being conducted at several sites:

- At BBN, Partridge and Jackson are exploring issues of programmability, data dictionaries, and authentication mechanisms, in the context of IP and to improve management and diagnostic capabilities.
- At the Georgia Institute of Technology, active network concepts are being applied to network congestion by allowing applications to request that specific node algorithms (e.g., lossless compression, selective discard, and transcoding) be invoked during periods of congestion [21].
- At the University of Kansas, Frost and Minden are considering the application of active technologies to rapidly deployable radio networks.
- At the University of Arizona, Peterson is developing "liquid" software, a suite of mobile code technologies that includes rapid compilation of intermediate code (i.e., at network link rates) [22].
- At the University of Cincinnati, Alexander is investigating techniques for the formal specification of network elements and behavior.

Summary

We realize that suggestions for software-intensive approaches to networking surface every ten years or so. For example, Zander [23] describes an experimental system in which packets of FORTH code were interpreted by network elements. Nonetheless, we are convinced that recent improvements in the safety and efficiency of active technologies, and the demand created by lead applications, present new research opportunities.

Active networks involve the synthesis and extension of programming language, operating systems, and networking expertise. We also anticipate changes to the organization of end system software — in place of protocol "stacks," applications may use protocol "components" that can be specialized and composed to perform application-specific functions [24]. This will lead to a massive increase in the degree and sophistication of network-based computation and address the mismatch between the rate at which user requirements change and the pace at which network infrastructure can be deployed.

Acknowledgments

This work has been influenced by discussions with a number of researchers, especially Deborah Estrin, David Farber, David Feldmeier, Henry Fuchs, Steve Garland, Carl Gunter, John Guttag, Frans Kaashoek, Butler Lampson, Paul Leach, Scott Nettles, Hilary Orman, Herb Schorr, Scott Shenker, Liuba Shrira, and Bob Sproull.

References

[1] M. Greenwald *et al.*, "Designing an Academic Firewall: Policy, Practice and Experience with SURF," *Proc. 1996 Symp. Network and Distributed Systems Security*, San Diego, CA, 1996.
[2] A. Chankhuntod, P. B. Danzig, and C. Neerdaels, A Hierarchical Internet Object Cache., *Proc. 1996 USENIX*, 1996.
[3] L. Kleinrock, "Nomadic Computing (Keynote Address)," *Proc. Int'l. Conf. Mobile Computing and Networking*, Berkeley, CA: ACM, 1995.
[4] H. Balakrishnan, *et al.*, "Improving TCP/IP Performance over Wireless Networks," *Proc. Int'l. Conf. Mobile Computing and Networking*, Berkeley, CA, 1995.
[5] E. Amir, S. McCanne, and H. Zhang, "An Application Level Video Gateway," *Proc. ACM Multimedia '95*, San Francisco, CA, 1995.
[6] M. T. Le, F. Burghardt, and J. Rabaey, "Software Architecture of the Infopad System," *Mobidata Wksp. Mobile and Wireless Info. Sys.*, New Brunswick, NJ, 1994.

[7] D. Tennenhouse *et al.*, "Virtual Infrastructure: Putting Information Infrastructure on the Technology Curve," *Comp. Networks and ISDN Sys.*, vol. 26, no. 13, Oct. 1996.
[8] N. Borenstein, "Email with a Mind of its Own: The Safe-Tcl Language for Enabled Mail," *Proc. IFIP Int'l. Conf.*, Barcelona, Spain, 1994.
[9] J. Gosling and H. McGilton, "The Java Language Environment: A White Paper," Sun Microsystems: Mountain View, CA, 1995.
[10] A. Adl-Tabatabai, *et al.*, "Efficient and Language-Independent Mobile Programs," *ACM SIGPLAN Conf. Programming Language Design and Implementation, PLDI '96*, Philadelphia: ACM, 1996.
[11] G. Necula, and P. Lee, "Safe Kernel Extensions Without Run-Time Checking," *2nd Symp. Op. Sys. Design and Implementation, OSDI '96*, Seattle, WA, 1996.
[12] D. Wetherall and D. Tennenhouse, "The ACTIVE IP Option," *Proc. 7th ACM SIGOPS Euro. Wksp.*, Connemara, Ireland, 1996.
[13] D. Mosberger and L. Peterson, "Making Paths Explicit in the Scout Operating System," Tech. Rep. 96-05, Dept. Comp. Sci., Univ. AZ,. 1996.
[14] D. R. Engler, M.F. Kaashoek, and J. O'Toole, Jr. "Exokernel: An Operating System Architecture for Application-Level Resource Management," *Proc. 15th ACM Symp. Op. Sys. Principles*, 1995.
[15] B. Bershad *et al.*, "Extensibility, Safety and Performance in the SPIN Operating System," 15th ACM Symp. Op. Sys. Principles, 1995.
[16] D. R. Engler, W. C. Hsieh, and M. F. Kaashoek, "'C: A Language for High-Level, Efficient, and Machine-Independent Dynamic Code Generation," *Proc. 23rd Annual ACM Symp. Principles of Programming Languages*, St. Petersburg, FL, 1996.
[17] D. Tennenhouse and D. Wetherall, "Towards an Active Network Architecture," *Proc. Multimedia Comp. and Networking, MMCN 96*, San Jose, CA, SPIE, Jan. 1996. A revised version of this paper appears in *Comp. Commun. Rev.*, vol. 26, no. 2, Apr. 1996.
[18] J. Smith *et al.*, "SwitchWare: Accelerating Network Evolution," Technical Report MS-CIS-96-38, CIS Dept., Univ. PA, 1996; also available at http://www.cis.upenn.edu/~jms/white-paper.ps.
[19] D. Feldmeier, A. McAuley, and J. Smith, "Protocol Boosters," submitted to *IEEE JSAC*, Jan. 1997.
[20] T. Yemini and S. da Silva, "Towards Programmable Networks," *Proc. FIP/IEEE Int'l. Wksp. on Distributed Systems*, Oct. 1996.
[21] B. Bhattacharjee, K. Calvert, and E. Zegura, "An Architecture for Active Networking," 1996, Technical Report GIT-CC-96-20, College of Comp., Georgia Inst. of Technology.
[22] J. Hartman et al., "Liquid Software: A New Paradigm for Networked Systems," 1996, Tech. Rep. 96-11, Dept. Comp. Science, Univ. AZ.
[23] J. Zander and R. Forchheimer, "SOFTNET — An Approach to High Level Packet Communication," *Proc. AMRAD Conf. 1983*, San Francisco, CA, 1983.
[24] D. D. Clark, and D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," *Proc. SIGCOMM '90*, 1990.

Biographies

DAVID L. TENNENHOUSE [M] is a principal research scientist at the Massachusetts Institute of Technology Laboratory for Computer Science and the Sloan School of Management. He is leader of the Telemedia, Networks and Systems Group, which is addressing "systems" issues arising at the confluence of broadband networks, advanced digital video, and distributed computing. David has a B.A.Sc. and an M.A.Sc in electrical engineering from the University of Toronto and a Ph.D. from the University of Cambridge. His e-mail address is dlt@lcs.mit.edu.

JONATHAN M. SMITH [SM] is an associate professor in the CIS Department of the University of Pennsylvania, where he is a member of the Distributed Systems Laboratory. He was co-principal Investigator on the NSF/DARPA-sponsored AURORA gigabit testbed project. He is an editor of *ACM/IEEE Transactions on Networking*. Jonathan previously worked for Boeing Aerospace, Bell Telephone Laboratories, and Bell Communications Research (Bellcore).

W. DAVID SINCOSKIE [F] is vice president of the Internet Research Laboratory at Bellcore. He supervises research on broadband services control, gigabit networking, feature interaction, IP over ATM, NSFNET, next-generation IP, and computer network management. He is a Senior Editor of *IEEE JSAC*, and was a member of the Internet Architecture Board from 1993 to 1995. David received his B.E.E. in 1975, M.E.E. in 1977, and Ph.D. (E.E.) in 1980 from the University of Delaware.

DAVID J. WETHERALL [StM] is a doctoral candidate at the MIT Laboratory for Computer Science. He is interested in networks, programming languages, and distributed systems. David joined the Laboratory after working at QPSX Communications, a high-speed networking company. He received his E.E and M.S. in computer science from MIT in 1995 and 1994, respectively, and his B.E. in electrical engineering from the University of Western Australia in 1989.

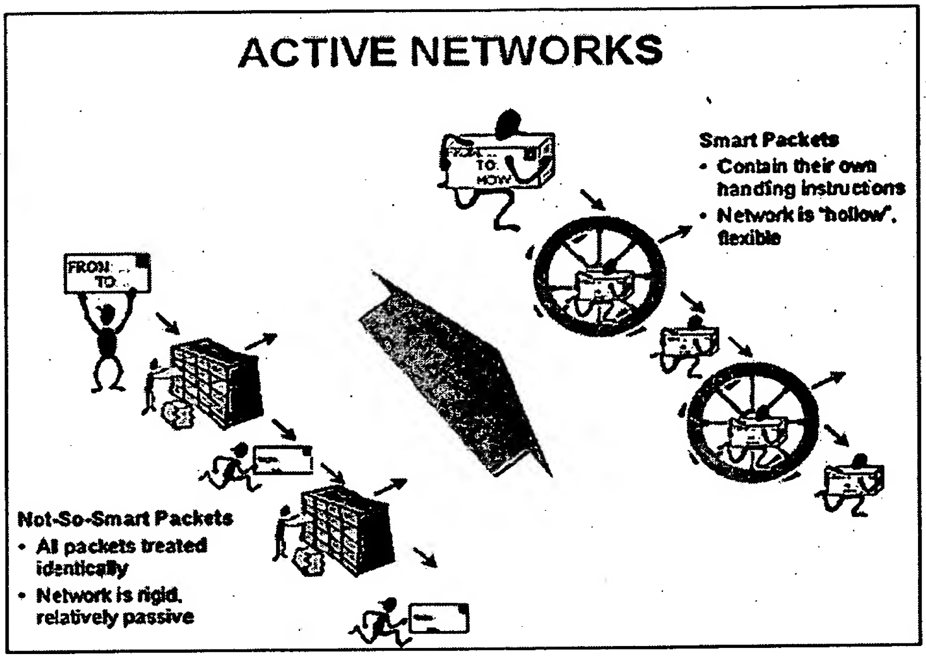
GARY J. MINDEN [M] is an associate professor in the EECS Department of the University of Kansas. He recently completed a tour at the Defense Advanced Research Projects Agency (DARPA) Information Technology Office. Gary's research interests are in the areas of large-scale distributed systems which encompass high-performance networks, computing systems, and distributed software systems. He received the B.S.E.E. degree in 1973 and the Ph.D. degree in 1982, both from the University of Kansas.

Home
What's New
Mission
Programs
Personnel
Solicitations
Contact

Active Networks

Mission

Networks that "turn on a dime."



Vision

The Active Networks program has the goal of producing a new networking platform, flexible and extensible at runtime to accommodate the rapid evolution and deployment of networking technologies and also to provide the increasingly sophisticated services demanded by defense applications.

The Active Nets architecture is based on a highly dynamic runtime environment that supports a finely tuned degree of control over network services. The packet itself is the basis for describing, provisioning, or tailoring resources to achieve the delivery and management requirements. A possible architecture utilizes a "Smart Packet" for the basic message unit on the Active Network; such a packet is an agent with the goal of delivering itself to its destination. The goal is expressed through a portion of the packet that describes its "method" -- a set of instructions that can be interpreted consistently by the Active Network nodes. The entire ensemble will be engineered to allow security, reliability, availability and quality of service to be tuned at multiple levels of granularity and under a wide range of conditions.

The evolution of defense networks by the injection of newly designed

services is needed in order to deploy new strategies or to tailor the infrastructure to immediate needs. The Active Network architecture supports this malleability as a first-class design goal, one that will reduce the time and cost of deploying new services. An additional dimension to network evolution will be the ability to support a multiplicity of network behaviors to be supported through the "virtualization" of the underlying infrastructure. Other crucial research topics include routing, resource allocation and network management services built with active network concepts.

There are crucial application areas that can benefit from the flexibility of the Active Network architecture, and breakthrough approaches will be sought. Areas of likely specialization include flexible, efficient, and secure protocols for: group communication strategies, scalable network management, quality of service management techniques, and radically more efficient routing protocols.

Goals

- **Quantifiable Improvement in Network Services**
- Audio/video synchronization and full-rate video over multicast
- Fewer retransmitted packets, 100% increase in useful data rate to end applications

- **Architecture Creates Solutions to Future DoD Needs**
- e.g., "addressless" networks, resource directed communication

- **Fault-Tolerance Mechanisms Based in Network**

- **Multi-Tiered Mobile Security**
- Authentication forms basis for dynamic access control
- Separate traffic and administrative functions based on types and policy

Challenges

Composite Protocols

- SmartPacket processing must be efficient, secure and survivable

Enhanced Network Services

- Quickly and safely deploy new services
- Achieve widespread use without need for standardization process
- Upgrade crucial network services to keep pace with network complexity (size, speed, variety)
- Develop new strategies for routing and service provisioning in large networks that have overlapping topologies and mobility requirements

Contact: ATO Security@darpa.mil

Active Networks - Smart Packets

Static Packets: Network elements constrained to simple functions